Behavioral Consistency for B2B Process Integration

Gero Decker and Mathias Weske

Hasso-Plattner-Institute, University of Potsdam, Germany
{gero.decker,mathias.weske}@hpi.uni-potsdam.de

Abstract. Interacting services are at the center of attention in business-to-business (B2B) process integration scenarios. Global interaction models specify the interaction behavior of each service and serve as contractual basis for the collaboration. Consequently, service implementations have to be consistent with the specifications. Consistency checking ensures that an implemented service is compatible with other services, i.e. that it can interact successfully with them. This is important in order to avoid deadlocks and guarantee proper termination of a collaboration. Different notions of compatibility between interacting services and consistency between specification and implementation are available but they are typically discussed independently from each other. This paper presents a unifying framework for compatibility and consistency and shows how these two notions relate to one another. Criteria for an optimal consistency relation with respect to a given compatibility relation are presented. Based on these criteria weak bi-simulation is evaluated.

1 Introduction

In the case of business-to-business (B2B) process integration different business partners interact with each other to reach a common goal. Each partner exposes its communication behavior as services that exchange business documents with the other partners' services in a certain order. Choreography languages such as WS-CDL ([5]) and Let's Dance ([15]) were put forward for capturing the interaction behavior from a global perspective. The interaction behavior is represented by interaction models, which serve as contractual basis for the collaboration between the partners. Interface processes, i.e. the behavioral specifications for the individual partners, can be generated (cf. [16]). These interface processes are the starting point for implementing new services or for adapting existing ones e.g. using BPEL ([1]). Consequently, the behavior of a service has to be consistent with the specified interface process. Such a consistency relation should ensure that an implemented service is in fact compatible with the partners' services without needing to check compatibility between the actual implementations. The latter is not desired since internal process details should not be revealed. Furthermore, if there is a large number of interacting partners, the number of possible combinations of partners that have to be checked for compatibility is so high that the testing phase becomes complex in itself.

J. Krogstie, A.L. Opdahl, and G. Sindre (Eds.): CAiSE 2007, LNCS 4495, pp. 81–95, 2007.

While different consistency relations have been reported in the literature, overarching criteria for evaluating consistency relations for specific purposes has not been proposed yet. This paper argues that a consistency relation can be checked for suitability with respect to a specific compatibility relation. E.g. the compatibility relation could allow that certain interactions never happen or that messages sent by one service are ignored by another one. Having chosen a suitable compatibility relation for a given context, we provide the criteria to decide whether a given consistency relation is optimal or not.

Existing notions of compatibility and consistency are discussed in the next section. Section 4 provides a formal definition of what an optimal consistency relation is with respect to a given compatibility notion. Section 5 elaborates on possible refinements from a process specification to an implementation. Since weak bi-simulation is a common formal basis for consistency checking, we will investigate in Section 6 whether it is optimal for the selected compatibility notions. Section 7 concludes and gives an outlook to future work.

2 Compatibility and Consistency in B2B Scenarios

Compatibility is the ability of a set of interconnected services to interact successfully. Consistency between a service implementation and a service specification is given if the implementation is valid with respect to the specification. In the literature also other names such as process inheritance are used as synonyms for consistency (cf. [4], [2]).

This section will further explain the need for compatibility and behavioral consistency using a B2B scenario. Figure 1 gives an overview over the partners in that scenario: A buyer (e.g., car manufacturer) uses reverse auctioning for procuring specially designed components. In order to get help with selecting the right suppliers and organizing and managing the auction, the buyer outsources these activities to an auctioning service. The auctioning service advertises the auction, before different suppliers can request the permission to participate in it. The suppliers determine the shipper that would deliver the components to the buyer or provide a list of shippers with different transport costs and quality levels, where the buyer can choose from. Once the auction has started, the suppliers can bid for the lowest price. At the end, the buyer selects the supplier according to the lowest bid or according to other criteria. After the auction is over, the auctioning service has to be paid for and shipment details are dealt with. Finally, the components are delivered to the buyer and are paid for.

It is obvious that there can be several suppliers, auctioning services, shippers and buyers. Therefore, different combinations of participants must be able to interact successfully. Unsuccessful interaction behavior could arise e.g., if different message formats are used in the collaboration and one participant does not understand the message content sent by other participants. Another source of incompatibility, which we will mainly focus on, is behavioral incompatibility. Imagine that a participant expects a notification at some point in a process before it can proceed and none of the other participants ever sends such a notification.

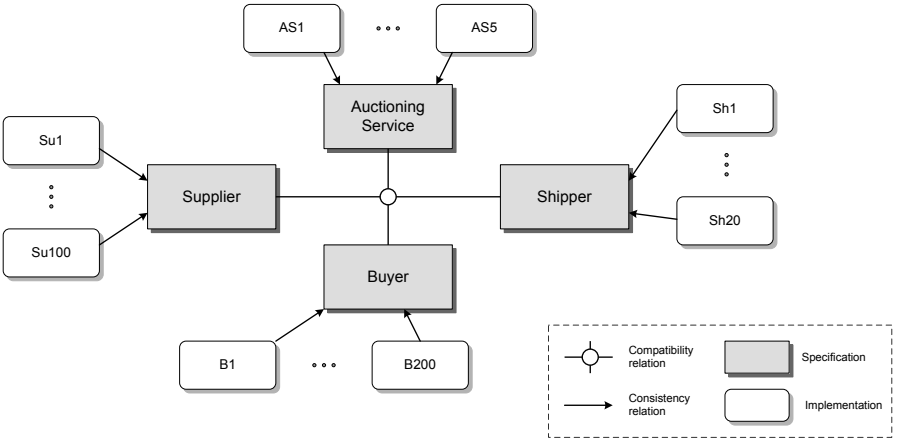

Fig. 1. Participants and roles in a reverse auctioning scenario

We call this situation a *deadlock*. In order to avoid deadlock situations and to ensure interoperability, the participants can agree on a certain desired interaction behavior. The behavioral constraints between message exchanges would be captured from the perspective of an ideal observer and the constraints for the communication behavior of every participant (the interface process) could be derived from such an interaction model (the choreography). This specification then serves as contractual basis for the collaboration and violations of the interaction contract could have legal consequences.

Figure 2 depicts a part of the collaboration specification where suppliers can request permission to the auction. We see that the roles supplier, auctioning service and seller take part in this collaboration. For each of these roles an interface process is given in the form of a Petri net. The places on the border of the dashed rectangle depict the structural interface of each role, i.e. the types of messages a role can potentially send or receive. The control flow between the communication actions constrains the execution. A "?" symbolizes a receive action and "!" a send action.

The supplier places a participation request at the auctioning service. The service formulates a recommendation whether to accept this supplier or not. This recommendation is normally based on previous experience with the supplier or legal requirements. The auctioning service sends the recommendation to the buyer. The buyer is not bound to this recommendation and can freely chose whether to accept or reject the supplier. Finally, the auctioning service forwards the decision of the buyer on to the supplier.

This specification does not tell the individual participants how their internal behavior should look like. The auctioning service could, for instance, lookup historical data about the supplier before coming up with a recommendation; also the buyer could have an internal decision making process possibly spanning different organizational units. No matter how the internal processes look like, we are concerned that the different participants successfully collaborate, i.e.

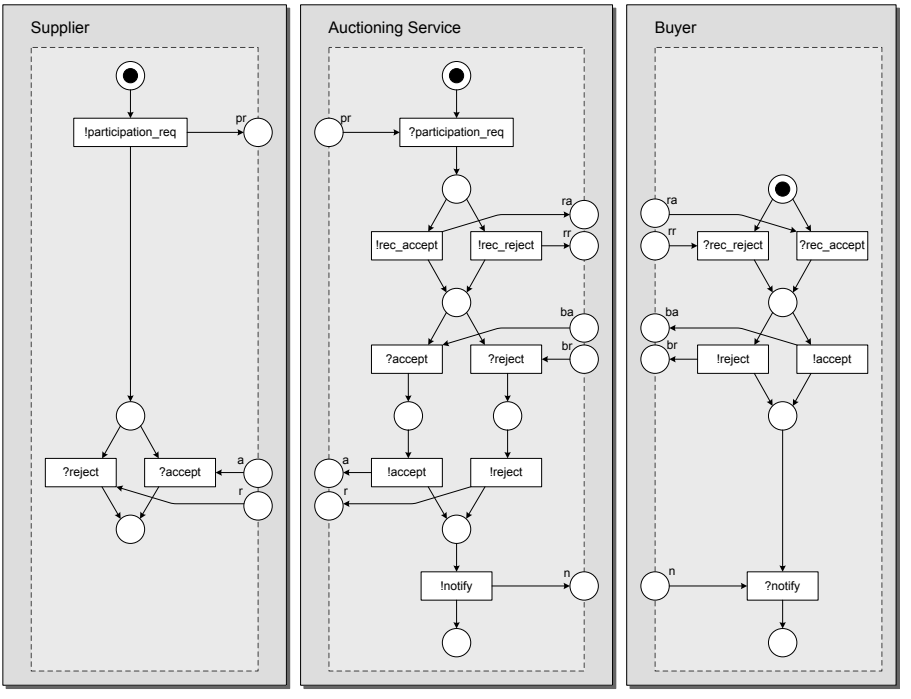

Fig. 2. Interface processes: Getting a participation permission

that the implementations are compatible. Ensuring compatibility is a challenging and cumbersome task when dealing with a large number of participants in this auctioning scenario, involving e.g., 100 suppliers, 20 shippers, 5 auctioning services and 200 buyers.

A remedy for this situation is the notion of consistency between interface and executable processes. The interface processes are the specifications for the different roles. Consistency between an interface process and the actual process implementation should ensure that the given implementation can interact with implementations for the other roles (provided that they in turn are consistent with their respective interface process). That way, we can locally check whether or not a participant should be allowed to be involved in the collaboration scenario. Compatibility between different implementations does not need to be checked any more.

3 Compatibility and Consistency Notions

In recent years there has been extensive work on different compatibility notions for interacting processes. This section compares four different notions, namely the compatibility notion by Martens [6], the compatibility notion by Canal et al. [4], interaction soundness by Puhlmann et al. [10] and the well-communicating

requirement used in the operating guidelines approach by Massuthe et al. [8,9]. Furthermore, this section will present different existing consistency notions.

Compatibility. First, we can distinguish between structural compatibility and behavioral compatibility. Structural compatibility like presented as "syntactic compatibility" in [6] demands that for every message that can be sent, the corresponding interaction partner must be able to receive it. Futhermore, for every message that can be received the corresponding partner must be able to send such a message. I.e. in the case of web service architectures the receiving service must have a matching operation for every outgoing SOAP message of the sending service, and for every operation a sending service must be able to send a corresponding message. We call this notion *strong structural compatibility*. In other compatibility notions, e.g. the well-communicating requirement, strong structural compatibility is not required. This acknowledges the fact that if a service provides a certain operation, the partners do not necessarily need to invoke this operation. However, it is still required that for every message sent there must be a corresponding operation. We call this *weak structural compatibility*. One could also think of examples where even weak structural compatibility is too restrictive: a middleware platform might be configurable in such a way that unprocessable messages are simply ignored. E.g. notifications that are not necessarily needed might not be received in some process. For such scenarios we introduce the notion of *minimal structural compatibility*. Minimal structural compatibility requires that there is at least one potential message send with a corresponding message receive by another participant.

Figure 3 presents three alternative service implementations for the buyer in our reverse auctioning example. We see that internal actions were added in the case of $B1$ and $B3$ (e.g. "Add to blacklist" and "Store decision"). $B1$ is structurally equivalent to the buyer in Figure 2 but has a different control flow structure. This alternative has strong structural compatibility with the Supplier and Auctioning Service, since every message sent can be received and for every message that can be received there is a message sent. This applies for both incoming and outgoing messages. $B2$ has only weak structural compatibility, since the Auctioning Service could receive a *reject* message from the buyer but the buyer never sends one. $B3$ does not have weak structural compatibility with the other participants: the notification sent by the Auctioning Service cannot be received by the buyer. However, since there are messages sent that can be received by the buyer, minimal structural compatibility is still given.

In contrast to structural compatibility, behavioral compatibility considers behavioral dependencies, i.e. control flow, between different message exchanges within one conversation. In most approaches the interface processes of the interacting partners are interconnected and reasoning is done on the resulting global process.

Martens bases his compatibility notion on interconnected workflow modules and requires strong structural compatibility. These modules are Petri nets with input, output and internal places (like the examples in Figures 2 and 3). When

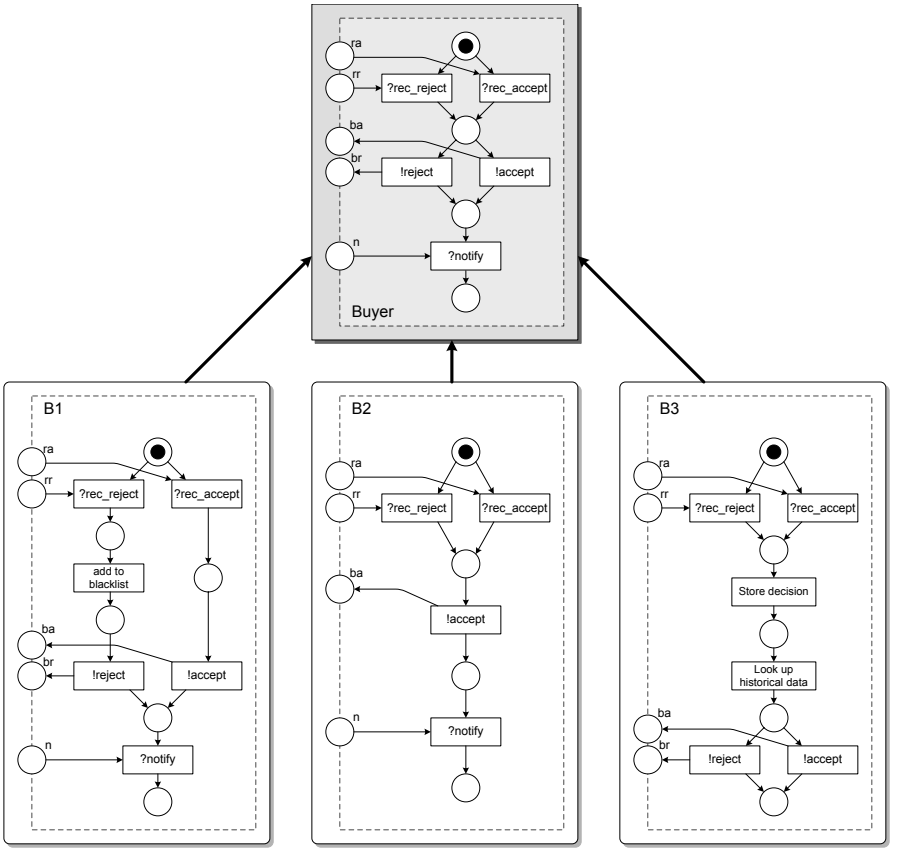

Fig. 3. Alternative implementations for the buyer

composing them, corresponding input and output places of interacting processes are merged and a global initial place and a global final place are added. Martens defines "weak soundness" on the global process, requiring that the final marking must always be reachable. This ensures that the global process is free of deadlocks and livelocks.

Canal et al. have also defined a compatibility notion for interacting π-processes. A main advantage of using π-calculus is the availability of *link passing mobility*. I.e. communication channels between interacting processes do not need to be statically defined but can be established at runtime. In real world settings this is called dynamic binding. E.g. a service broker passes the reference to a provided service on to a service consumer who can then use the service. π-interactions are atomic, i.e. sending and receiving of messages happen at the same time. Therefore, it is not possible that one π-process sends a message which is not consumed by the other. The compatibility notion by Canal et al. requires that both processes complete, i.e. that no more sending or receiving action is left

to be performed. A major drawback of the given compatibility notion is that it is defined for bi-lateral settings only.

Interaction soundness is based on "lazy soundness" of the global process. It is required that the process always completes, while some activities are still allowed to run even after completion. Considering these "lazy activities" is essential for coping with advanced control flow constructs such as Discrimators (cf. [12]) but leads to the fact that livelocks cannot be detected in some situations. Interaction soundness is defined for a combination of a service and its environment. We find a mixture of strong and minimal structural compatibility between the service and its environment: The environment must be able to send and receive all those kind of messages that the service can receive or send. Therefore, there must be strong structural compatibility in one direction. However, the service is not required to send and receive all those kind of messages that the environment is able to receive or send. Interaction soundness is defined on π-calculus. Therefore, it can also deal with link passing mobility.

The operating guidelines approach to checking compatibility [8,9] is different to the three previous approaches in that it does not reason on interconnected interface processes. Rather an "operating guideline" can be generated for an interface process which includes all valid interaction behavior that respects the well-communicating requirement. This requirement includes weak structural compatibility and the absence of deadlocks and livelocks. Operating guidelines are annotated state machines and represent the most permissive interaction behavior for the interaction partners. The interface processes of the partners (also given as state machines) must then be sub state machines of the most permissive behavior. The main motivation behind the operating guidelines approach is to reach a smaller computational complexity for compatibility checking. A current limitation of operating guidelines is that only acyclic processes are allowed.

Consistency. There has been quite some research work on consistency relations between specified interface processes and process implementations comparing their observable behavior. Basten et al. introduce different notions of process inheritance in [2], namely protocol inheritance, projection inheritance, protocol/projection inheritance and life-cycle inheritance. In order to determine whether an implementation is a subclass of a given specification, encapsulation and abstraction mechanisms are employed. Encapsulation deletes additional activities from the implementation before comparing it to the specification, while abstraction re-labels certain activities as τ-actions, i.e. they are not considered in the weak bi-simulation relation. Once encapsulation and abstraction is applied, branching bi-simulation is used to compare the two process definitions. Bi-simulation relations were defined for different formalisms. These results are used in the public to private approach reported in [13], where participants agree on a global interaction model and a partitioning of this model to participants. Using inheritance mechanisms, each partner can implement an arbitrary subclass of their public process as a local, private implementation. The inheritance rules

make sure that the private implementations satisfy the interaction constraints defined in the public model.

Other examples for (bi-)simulation relations are weak open (bi-)simulation for π-calculus by Sangiorgi [11] and branching bi-simulation for Petri nets by van Glabbeek and Weijland [14]. Bi-simulation in general will be discussed in section 6. Busi et al. have introduced their own calculi for choreographies and orchestrations in [3]. Consistency between orchestration and the specified behavior, which is given in the choreography, is shown through a bi-simulation-like relation, which is also defined by the authors. Martens presents a consistency relation in [7] where the implementation must accept at least those messages specified and must produce at most those messages specified.

4 Optimal Consistency Relations

The previous section recapitulates different notions of compatibility and consistency. Consistency is dependant on compatibility and should go hand in hand with it. Therefore, this section introduces a means to judge whether or not a given consistency relation is suited for a given compatibility notion. The following two requirements for consistency relations can be identified:

1. *A consistency relation should ensure compatibility.* If a set of specified service definitions are compatible and a set of implementations are consistent with these specifications then this set must also be compatible. Figure 4 illustrates this. If A_{spec}, B_{spec} and C_{spec} are compatible and A_{impl} is consistent with A_{spec}, B_{impl} with B_{spec} and C_{impl} with C_{spec}, then A_{impl}, B_{impl} and C_{impl} must also be compatible.
2. *A consistency relation should not be too restrictive.* The consistency relation should be maximal, i.e., every possible extension to the consistency relation must result in the violation of the previous requirement.

In the remainder of this section the abovementioned requirements are formalized. To allow for reusing the definitions for arbitrary compatibility and consistency notions, the formalization is independent from a particular formalism (such as Petri nets or π-calculus). We introduce as follows:

- *S is a set of service definitions*,
- *$C \subseteq \wp(S)$ is the set of all compatible combinations of service definitions* and
- *$\succeq \subseteq S \times S$ is a binary relation on S where $s_{impl} \succeq s_{spec}$ denotes that service definition s_{impl} is consistent with service definition s_{spec}.*

In the example shown in Figure 4, all spefications and implementations are service definitions: A_{spec}, B_{spec}, C_{spec}, A_{impl}, B_{impl}, $C_{impl} \in S$, the set of specifications and the set of implementations are compatible, respectively: $\{A_{spec}, B_{spec}, C_{spec}\}$, $\{A_{impl}, B_{impl}, C_{impl}\} \in C$ and the implementations are consistent with their respective specifications: (A_{spec}, A_{impl}), (B_{spec}, B_{impl}), $(C_{spec}, C_{impl}) \in \succeq$.

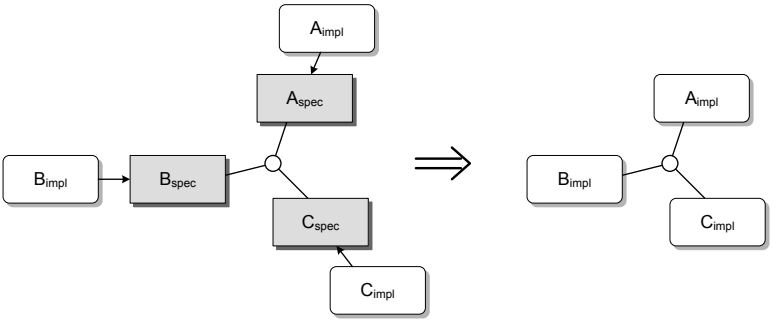

Fig. 4. The consistency relation must respect the compatibility notion

An auxiliary relation is introduced, $\succeq' := \{(c_1, c_2) \in \wp(S) \times \wp(S) \mid \forall s_1 \in c_1\ [\exists s_2 \in c_2\ (s_1 \succeq s_2)] \wedge \forall s_2 \in c_2\ [\exists s_1 \in c_1\ (s_1 \succeq s_2)]$: All service definitions s_1 in one set are consistent with at least one service definition s_2 in the other set and there is no s_2 without at least one s_1 that is consistent with it.

Based on these definitions, we can formalize the two requirements for optimal consistency relations: A consistency relation $\succeq$ is optimal for a compatibility notion C if and only if

1. $\succeq$ *respects* C, i.e. $\forall c_1, c_2 \in \wp(S)\ [(c_2 \in C \wedge c_1 \succeq' c_2) \Rightarrow c_1 \in C]$
2. $\forall (s_1, s_2) \in S \times S\ [\neg(s_1 \succeq s_2) \Rightarrow \neg((\succeq \cup \{(s_1, s_2)\})\ respects\ C)]$

The first line states that a combination of service definitions c_1 must be compatible, if c_1 is consistent with a combination of service definitions c_2 that are compatible. We have introduced the relation $respects \subseteq (S \times S) \times \wp(S)$ indicating which consistency relation respects which compatibility notion. The second line states that adding a new tuple (s_1, s_2) of service definitions to the consistency relation must result in breaking the first requirement. I.e. we must not disallow any implementation that would successfully interact with the other allowed implementations.

5 Process Refinement Categories

The definition of optimal consistency relations in the previous section allows to decide a true/false decision about the suitability of a given consistency relation. In this section we want to describe some typical process refinements, i.e. differences between specified interface processes and process implementations. An optimal consistency relation is expected to support all these process refinements, provided that the compatibility notion is permissive enough. The list of refinement categories allows to compare compatibility and consistency notions with respect to engineering needs. If a consistency relation is not optimal for a given compatibility notion it probably provides less support for at least one of the categories. Unlike the requirements for optimal consistency relations we do not provide formal definitions of the refinements.

1. **Addition of internal actions.** Specified interface processes indicate what interaction behavior other participants can expect from an organization. On the other hand, a process implementation covers all internal activities and dependencies that are present within the organization. In other words, the specified interface process constrains the interaction behavior of an organization while the implementation contains all details for actually executing the process. The implementation also shows interdependencies with other processes running within the organization. Therefore, additional internal activities that are not visible outside the organization can be found in the process implementation.
2. **Addition of communication actions.** The process implementation sometimes needs to be able to comply to the constraints given in different interface processes. E.g. there might be different interaction contracts with upstream and downstream partners in a supply chain scenario. In order to cope with such a situation the process implementation contains more communication actions with partners than specified in one interface process.
3. **Deciding choices at design-time.** If a partner is allowed to do choices, e.g. an organizational unit can decide whether to send an accept or reject message (cf. the buyer in Figure 3), the specification indicates the latest moment where the choice can be made. However, an organization might decide that always the same branch is taken. E.g. the buyer in the example of the previous section might decide to always accept a supplier (cf. B2 in Figure 3). Therefore, the choice is already done at configuration time of the partner's system, i.e. design-time of the process implementation.
4. **Removal of communication actions.** When deciding at design-time that particular branches should be taken then this results in removing communication actions from the process that are part of the other branches. This has an influence on structural compatibility if all communication actions for a particular message type are deleted. Another reason for deleting receive actions could be the knowledge that the sending party has done a design-time decision never to take a certain branch. However, it can be argued that such an implementation should not be allowed by the consistency relation because such an agreement between the two partners is not reflected in the specification.
5. **Deciding choices earlier at runtime.** It is also imaginable that a choice is done sometime earlier in the process. E.g. an organizational unit decides dependending on what message comes in sometimes earlier in the process. B1 in Figure 3 sends an accept or reject message if a corresponding recommendation comes in. Therefore, the choice whether to send an accept or reject message is not done right before such a message is to be sent, as it is the case in the specification, but the choice is rather done as soon as a recommendation is received. In other cases, deciding choices earlier at runtime does not have any visible effect for the outside world.
6. **Sequentialization of communication actions.** Ideally, specified interface processes do not make any restrictions on the ordering of message production and consumption if not absolutely necessary. E.g. once a supplier is selected,

a buyer is required to initiate the payment for the auctioning service and send delivery details to the designated shipper. Although there is no constraint in what order the buyer has to send the two messages, the process implementation might sequentialize it. E.g. the shipper is always notified as soon as possible while the payment is delayed for a while.

7. **Reordering of communication actions.** Consider a similar scenario like the previous one: it is specified that a seller first receives payment details before he receives the delivery details from the buyer. Assuming asynchronous communication, it might be allowed that the seller can reorder the consumption of the two incoming messages for process optimization purposes: He first processes the delivery details and initiates transport before he processes the payment details.

6 Assessment of Bi-simulation for Consistency Checking

Section 3 has shown that weak bi-simulation is the basis for several consistency relations. The main idea behind weak bi-simulation is that a process A can simulate the communication behavior of process B and vice-versa, while internal actions are not considered. Therefore, if A is capable of doing some communication action c then B must also be capable of doing c and again vice-versa. In the case of consistency checking we can therefore compare a specified interface process with a process implementation in terms of bi-similarity.

Since two bi-simulation related processes A and B show equivalent communication behavior, it is easy to see that bi-simulation *respects* a wide range of compatibility notions. The first criterion for an optimal consistency relation is therefore given. The remainder of this section investigates whether bi-simulation is too restrictive, i.e. if the second requirement for an optimal relation is met.

Consider an example similar to the B2B scenario introduced in Section 2. In Figure 5 processes P_1 and P_2 are depicted having the same structural interface to the environment: Messages of type a and b can be received, and messages of type c and d can be produced. Due to this structural equivalence, a and b have the same degree of structural compatibility with any given environment. However, these processes show different behavior. In terms of combinations of communication actions that are performed within one process instance, process P_1 allows $?a.!c$, $?a.!d$, $?b.!c!$ and $?b.!d$. On the other hand, P_2 only allows $?a.!c$ and $?b.!d$. In this example, P_1 can simulate all behavior of P_2 but P_2 cannot simulate all behavior of P_1. Therefore, P_1 and P_2 are not bi-simulation related.

The second example (Figure 6) leads to a similar situation. In P_3 no ordering constraint between $!f$ and $!g$ is given, while in P_4 $!f$ always happens before $!g$. P_3 therefore allows $?e.!f.!g$ and $?e.!g.!f$, while P_4 only allows $?e.!f.!g$. P_3 and P_4 are not bi-simulation related.

According to the consistency notions in [2] and [3], which are based on bi-simulation, P_2 and P_4 would not be allowed as implementations for P_1 and P_3. However, P_2 would be compatible with all environments that P_1 is compatible with. This is due to the fact that the environment must be able to receive c and d.

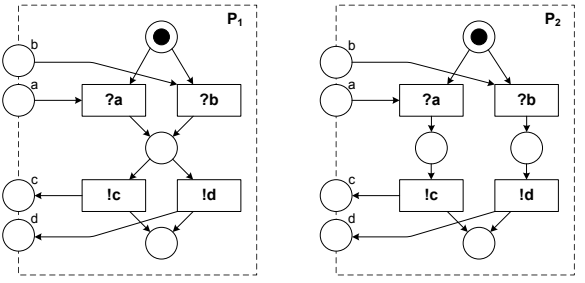

Fig. 5. P_1 and P_2 are not bi-simulation related

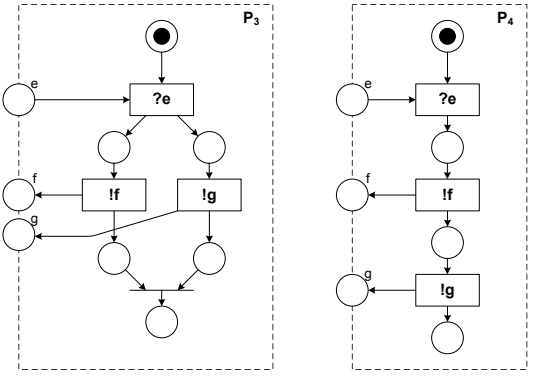

Fig. 6. P_3 and P_4 are not bi-simulation related

The environment is not allowed to do any assumptions about which message is to be received, otherwise it would be incompatible with P_1. For this reason, it is just a special case that in P_2 the choice for c or d is linked to the previously received message. In analogy to this, we know that P_4 would be compatible with all environments that P_3 is compatible with.

Therefore we can conclude that bi-simulation is too restrictive as compatibility notion for the compatibility notions presented in section 3, since adding the tuples (P_1, P_2) and (P_3, P_4) to the consistency relation would not result in not respecting any of the compatibility notions presented in section 3 any longer.

These examples also show that consistency relations do not need to be symmetric, i.e. that if a process A is consistent with process B, B does not need to be consistent with A. Assume a process X allowing the two combinations $!a.?c$ and $!b.?d$ would be compatible with P_2 but not with P_1: Assume that P_1 produces a message of type d after having received a message of type a, i.e. $?a.!d$, the conversation would deadlock since X would expect a message of type c which in turn would never be sent in that conversation. Figure 7 depicts this situation. Since all presented compatibility notions detect such simple potential deadlocks, incompatibility holds with respect to all these notions.

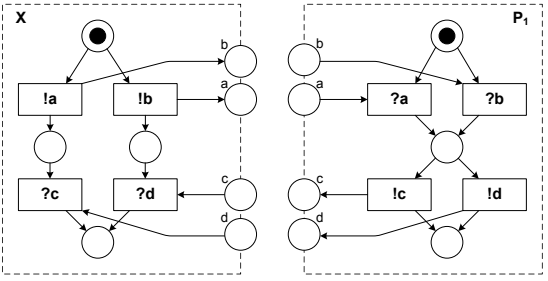

Fig. 7. X and P_1 are not compatible

The processes in Figures 5 and 6 are examples for the process refinement categories presented in the previous section. In P_2 the choice whether a message of type c or d is sent is made earlier than in P_1, namely already as soon as a message is consumed. P_4 is an example for the sequentialization of actions.

Let us now take a look at the other categories from Section 5. Weak bi-simulation directly supports the category *addition of internal activities*. The added activities are simply treated as τ-actions and are therefore ignored. For *addition of communication actions* it might be possible to re-label the added actions as τ-actions before testing for bi-similarity. This pre-processing step is suggested in [2]. However, it cannot be generally allowed. Especially if message exchanges for existing message types are added, compatibility with other processes might be affected. *Removing communication actions* is not supported through bi-simulation in the case of reachable communication actions. *Deciding choices at design-time* is not supported (assuming that it affects the communication behavior). When *deciding choices earlier at runtime* the communication behavior is not affected in many real-world cases. Therefore, we conclude that there is partial support for this category. *Sequentialization* and *reordering of communication actions* are not supported through bi-simulation.

Table 1 summarizes what categories of process refinements are supported through weak bi-simulation. A "+" denotes that there is full support, "+/–" partial support and "–" indicates no support (including the assumptions made in the previous paragraph). The table highlights that weak-simulation does not

Table 1. Common process refinements and support through weak bi-simulation

Process refinements	**Weak bi-simulation**
1. Addition of internal actions	+
2. Addition of communication actions	+/–
3. Deciding choices at design-time	–
4. Removal of communication actions	–
5. Deciding choices earlier at runtime	+/–
6. Sequentialization of communication actions	–
7. Reordering of communication actions	–

fully support a wide range of common process refinements and therefore has limited suitability.

7 Conclusion

This paper motivates the need for behavioral consistency checking in B2B process integration scenarios. Especially in choreography-driven settings such consistency is of key importance. We have introduced a unifying framework for behavioral compatibility and consistency of services. Two requirements for consistency relations have been introduced for classifying whether a consistency relation preserves compatibility and if it is too restrictive with respect to the compatibility relation. It was shown that interacting partners only need to agree on a suitable compatibility notion for their purposes and no further discussion about the consistency relation is required, since it can be determined whether or not a consistency relation is optimal for the chosen compatibility notion.

Furthermore, it was shown that classical weak bi-simulation relations do not fulfill the two requirements and refinement categories were highlighted that are fully, partially, or not supported by weak bi-simulation.

In future work, we are going to investigate other consistency relations with respect to corresponding compatibility notions. The consistency relation introduced by Martens ([7]) is promising. It might turn out to be optimal for his weak-soundness-based compatibility notion introduced in [6]. In addition we are going to define optimal consistency relations for selected compatibility notions.

References

1. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services, version 1.1. Technical report, OASIS, (May 2003). `http://www-106.ibm.com/developerworks/webservices/library/ws-bpel`
2. Basten, T., van der Aalst, W.M.P.: Inheritance of behavior. JLAP 47(2), 47–145 (2001)
3. Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., Zavattaro, G.: Choreography and Orchestration: A Synergic Approach for System Design. In: Proceedings 3rd International Conference on Service Oriented Computing (ICSOC, Amsterdam, The Netherlands, Springer (December 2005)
4. Canal, C., Pimentel, E., Troya, J.M.: Compatibility and inheritance in software architectures. Sci. Comput. Program. 41(2), 105–138 (2001)
5. Kavantzas, N., Burdett, D., Ritzinger, G., Lafon, Y.: Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation. Technical report,(November 2005) `http://www.w3.org/TR/ws-cdl-10`
6. Martens, A.: Analyzing Web Service based Business Processes. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, Springer, Heidelberg (2005)
7. Martens, A.: Consistency between Executable and Abstract Processes. In: Proceedings IEEE International Conference on e-Technology, e-Commerce, and e-Services (EEE 2005), Hong Kong, China, pp. 60–67. IEEE Computer Society Press, Los Alamitos (March 2005)

8. Massuthe, P., Reisig, W., Schmidt, K.: An Operating Guideline Approach to the SOA. Annals of Mathematics, Computing and Teleinformatics 1(3), 35–43 (2005)
9. Massuthe, P., Schmidt, K.: Operating guidelines - an automata-theoretic foundation for the service-oriented architecture. In: Proceedings Fifth International Conference on Quality Software (QSIC 2005), pp. 452–457. IEEE Computer Society Press, Washington, DC, USA (2005)
10. Puhlmann, F., Weske, M.: Interaction Soundness for Service Orchestrations. In: Dan, A., Lamersdorf, W. (eds.) Proceedings of the 4th International Conference on Service Oriented Computing ICSOC 2006, LNCS, vol. 4294, pp. 302–313. Springer, Heidelberg (2006)
11. Sangiorgi, D.: A Theory of Bisimulation for the pi-Calculus. Acta Informatica 16(33), 69–97 (1996)
12. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
13. van der Aalst, W.M.P., Weske, M.: The P2P Approach to Interorganizational Workflows. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 140–156. Springer, Heidelberg (2001)
14. van Glabbeek, R., Weijland, W.: Branching Time and Abstraction in Bisimulation Semantics. Journal of the ACM 43(3), 555–600 (1996)
15. Zaha, J.M., Barros, A., Dumas, M., ter Hofstede, A.: A Language for Service Behavior Modeling. In: Proceedings 14th International Conference on Cooperative Information Systems (CoopIS 2006), Montpellier, France, Springer, Heidelberg (November 2006)
16. Zaha, J.M., Dumas, M., ter Hofstede, A., Barros, A., Decker, G.: Service Interaction Modeling: Bridging Global and Local Views. In: Proceedings 10th IEEE International EDOC Conference (EDOC 2006), Hong Kong (October 2006)