Communication models for services

Niels Lohmann[1,2]

[1] Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, P. O. Box 513, 5600 MB Eindhoven, The Netherlands
[2] Universität Rostock, Institut für Informatik, 18051 Rostock, Germany
niels.lohmann@uni-rostock.de

Abstract. Communication is an essential aspect of services. Services do not only realize simple request-response scenarios, but increasingly implement complex and stateful communication protocols. Such a protocol specifies the order in which messages are sent and received by a service and is an essential part of a service description. Services are usually not executed in isolation, but as a collaboration which is composed of several services. The behavior of such a collaboration is not only determined by the communication protocol of each participating service, but also by the way messages are exchanged. A *communication model* specifies the properties of the message channels between the services and defines the way how messages are sent and received. This paper studies and classifies several dimensions of communication models and describes their impact to the behavior of service compositions.

1 Introduction

The paradigm of service orientation aims at replacing complex monolithic systems by a composition of several simpler, yet logically or geographically distributed components, called services. This distributed nature introduces new problems, because independent services need to collaborate to reach a common goal. One way to achieve correct collaboration is to offer *standardized* service descriptions. The most prominent example is the language WS-BPEL [3] which emerged as standard to specify executable Web services as well as to describe communication protocols on an abstract level. WS-BPEL received much attention from academia, and there exists a variety of formalizations of the communication protocol of a WS-BPEL process; van Breugel and Koshkina [7] present a survey of hundreds of papers. However, these formalizations usually focus on the execution order of the service's activities and provides an answer to the question *when* communication takes place, but give little details on the way *how* messages are exchanged.

Although a communication protocol specifies many dependencies between activities — be it internally or between one service's send activity and another service's receive activity — other details remain unspecified. Is the message exchange atomic, or is sending and receiving decoupled? Can the receiver block the sender? Can messages be buffered, and if yes, how many of them? Each of these questions is not addressed by the WS-BPEL specification. However, they still have an impact on the behavior of the overall collaboration.

We observed that — apart from a vague distinction between *synchronous* and *asynchronous* message exchange — there does not exist a common understanding how communication should be modeled. This in turn makes it hard to compare existing results on the formalization and analysis of services. To this end, this paper studies *communication models*. A communication model specifies the aspects of the message exchange such as those previously sketched. It may also include the occurrence of *faults*. A fault is an undesired and abnormal scenario (i. e., a buffer overflow) for that the subsequent behavior is unspecified. A communication model thereby can be seen as the missing piece to completely describe and reason about the behavior of a service composition. Note that communication models are not tied to any specific service description language or formalization.

This paper is not a survey on existing service description languages or service formalizations; we refer the interested reader to survey [11,16]. Actually, Kazhamiakin et al. [16] already study a wide spectrum of communication models and investigate questions related to their expressive power. Compared with that approach, this paper aims at providing an intuitive classification of communication models which is not guided by expressive power or a concrete formalization; the dimensions studied in Sect. 2 are more general than those of Kazhamiakin et al. Furthermore, this paper does not discuss the suitability of certain communication models nor tries to compare formalisms or service description languages with respect to their assumed communication model(s) as it is done by approaches such as the *service interaction patterns* [4]. Instead, we want to highlight certain effects and consequences which arise with the choice of a certain communication model. We do this by examining in Sect. 3 the service's behavior from the point of view of partner services. Section 4 concludes the paper and lists several directions of future work.

2 Dimensions of communication models

The most apparent impact of the choice of a communication model becomes visible in the level of abstraction of the message buffer; that is, the infrastructure which transfers messages between two services. In case of *synchronous communication*, message exchange is assumed to be instantaneous: Messages are not buffered and no intermediate state in which a sent message is pending is modeled. Thereby, the meaning of the term "synchronous" is closer to "isochronous" rather than "synchronization", because the latter can also be realized with two messages modeling a handshake.

On the other hand, sending of a message can be decoupled from receiving; that is, sent messages are buffered until they are received and, as a consequence, communication is *asynchronous*. Even though considering intermediate states yields in an increased complexity, asynchronous communication allows for more efficient communication, because sender and receiver do not need to constantly synchronize, but can be executed more autonomously. Thereby, asynchronous communication naturally supports the distributed setting of services.

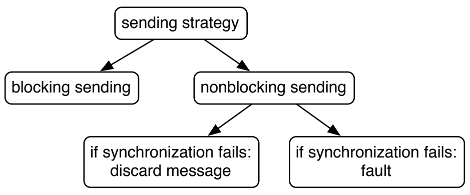
sending strategy
blocking sending
nonblocking sending
if synchronization fails: discard message
if synchronization fails: fault

Fig. 1: Dimensions of synchronous communication models.

2.1 Synchronous message exchange

In the simplest communication model, message exchange is atomic. The sender waits until the receiver is ready for the message exchange, and then they synchronize by executing the sending and the receiving activity simultaneously. In this setting, the state in which the message is sent, but not yet received, is not modeled: the message channel is abstracted from. Consequently, the message exchange is executed in an all-or-nothing fashion. It is notable that the direction of the message exchange is irrelevant from a technical point of view, because there exists no distinguished initiator.

Variants of this synchronous communication loosen this symmetry between sender and receiver and allow for nonblocking execution of the sending activity. In case the sending activity can be executed independently from the receiving activity, the setting in which the receiving activity is not ready for execution (i. e., the synchronization fails) needs to be specified. Then, the message can be either discarded or a fault occurs. The former scenario is motivated by signal nets [17] which introduce one-sided synchronization of modules in the setting of control engineering. Desel [11] describes an application to services.

Figure 1 provides an overview of the different dimensions of synchronous communication models. The most prominent service model using synchronous communication is the "*Roman model*" [5] in which message transfer is specified by a synchronous communication model that assumes blocking sending. The same communication model is also used by earlier formalizations of interaction such as *I/O automata* [19] or *interface automata* [8].

2.2 Asynchronous message exchange

In contrast to synchronous message exchange, asynchronous communication models refine the message exchange and decouple the sending of a message from its receiving. This is usually motivated by performance issues, and the fact that the actual moment a message is received should be modeled independently of the moment of the sending. Consequently, the state in which the message is in transit (i. e., already sent but not yet received) is explicitly modeled. Consequently, an asynchronous communication model not only needs to specify the characteristics of the sending of a message, but also its transfer and its receipt. These characteristics can be grouped into properties of the employed message buffer (including how it can be accessed by the receiving activity) and sending strategies.

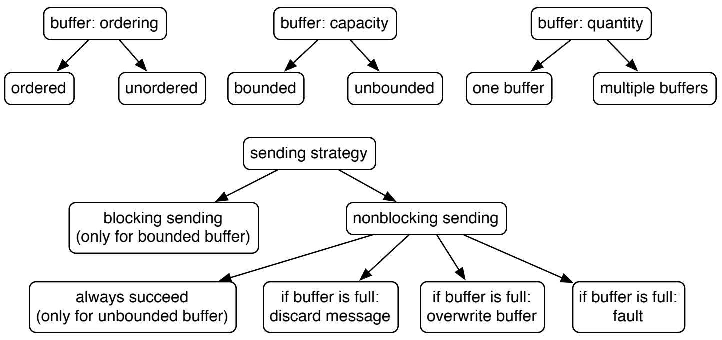

Fig. 2: Dimensions of asynchronous communication models.

Message buffers. An asynchronous communication model needs to specify three aspects of message buffers: their capacity, their organization, and their quantity.

The *capacity* of a message buffer determines the maximal number of messages which can be simultaneously buffered. In literature, the case in which a message buffer is unbounded is often considered. This absence of a capacity is typically motivated as a proper abstraction from a concrete, but unknown upper bound. Actually, the determination of the maximal capacity is not trivial, because it is not only influenced by the service under consideration, but also by those services which are composed to it. Nevertheless, the need of a finite and fixed capacity is motivated by the middleware which realizes the communication of services in reality. If the capacity is not known in advance, it may be approximated using capacity considerations or static analysis techniques, or it is chosen sufficiently large.

The second aspect specifies how buffered messages are *organized*. As Kazhamiakin et al. [16], we distinguish ordered and unordered buffers. In the first case, the order in which messages are added to the buffer is preserved, and the receiver only has access to the "oldest" (i. e., earliest sent) element. Ordered buffers are usually modeled by FIFO queues. In the second case, messages are buffered unordered, modeling a channel in which messages may overtake one another. Consequently, the receiver has access to all buffered messages. This scenario over-approximates any transfer delay an asynchronous medium can introduce.

Finally, communication can be organized using *multiple* buffers. In the setting of ordered buffers, this allows the receiver to access more messages and hence may enable more message-receiving activities compared with the case where only one message can be accessed. For the unordered case, multiple buffers do not introduce additional behavior. In addition, by organizing each message in a separate ordered buffer, unordered buffers can be simulated.

Sending strategies. Similar to synchronous message exchange, different parameters influence the way sending activities are executed. For unbounded buffers, a nonblocking sending is the simplest case. In case a buffer capacity is assumed, the situation in which the buffer is full needs to be specified. Either, the message cannot be sent (blocking) or sending is nonblocking and the message is discarded, a buffered message is overwritten, or an error occurs.

Figure 2 diagrams the four dimensions of asynchronous communication models. Obviously, the buffer capacity and the buffer quantity need to be explicitly specified further than "bounded" and "multiple", respectively.

As an example from literature, *open nets* (previously called *open workflow nets*) use interface places to model a message buffer. Using Petri net places yields an asynchronous communication model with a single unordered buffer. The model itself does not pose a capacity of the buffer and hence open nets impose an "always succeed" sending strategy. Similarly, *concurrent automata* [2] assume a multiset as channel model, yielding a single unordered buffer. *Communicating FSM* [6] employ unbounded ordered buffers (i. e., FIFO queues) to model communication. Hence, sending is nonblocking. In case a service communicates with more than one other service, one FIFO queue is assumed for each pair of services.

3 Impact of communication models

The previous section sketched different dimension of communication models. In this section, we demonstrate how the choice of a communication model has an impact on the controllability of a service. A service is *controllable* [22] if there exists a partner service such that their composition can always eventually reach a final state in which the message channels (if modeled) are empty. To visualize service models, we use BPMN [21] as graphical notation.

As a first example, consider the service in Fig. 3(a). It controllable if we assume synchronous communication. An asynchronous communication model would need to specify a buffer capacity of at least 2 in case messages are not discarded, because a communication partner cannot observe the receipt of the A message. Figure 3(b) shows compatible partner for bounded channels with a "discard" strategy.

Figure 3(c) demonstrates the impact of ordering of buffered messages: this service can only be controlled with synchronous communication or ordered buffers, because reordering of messages A and B results in a situation in which a partner needs to "guess" whether to send a C or D message, and any guess could yield unreceived messages. A similar situation occurs in the interaction with the service in Fig. 3(d). This service cannot be controlled with an asynchronous communication model at all, because the result internal choice (modeled by an XOR gateway) cannot observed by a communication partner. In contrast, a partner with a synchronous communication model with blocking sending, however, can control the service.

Figure 3(e) shows an example of the impact of the buffer quantity. If we assume a single ordered buffer, this service is controllable, for instance by the service in Fig. 3(f). If we assume the transfer of message E takes very long (e. g., in case of a large video file), we cannot speed up the service composition by sending it concurrently to other messages as it is done in Fig. 3(g) which would

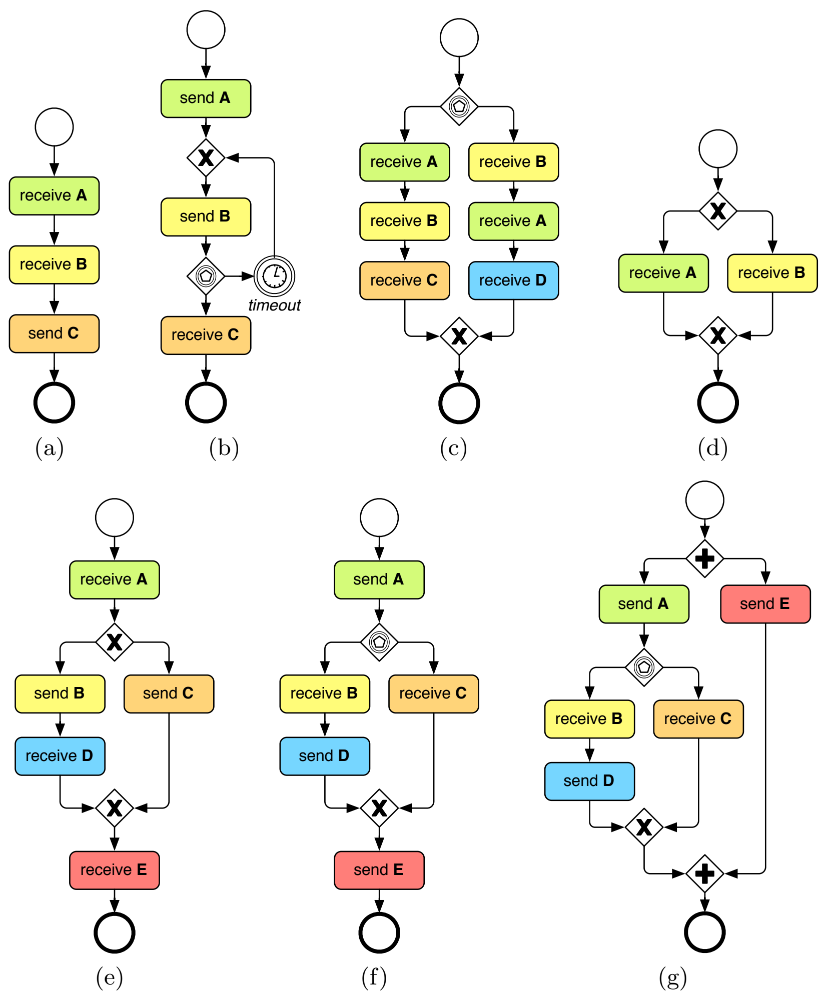

Fig. 3: Impact of communication models to service behavior.

be a correct partner in case we assume a distinct buffer for message E or an unordered buffer.

The presented examples show that the communication model has an impact to the overall behavior of a service. In particular, message buffers may introduce "new" behavior by enabling message-receiving activities. This additional behavior may yield in concurrent and more flexible service compositions on the one hand, but also in undesired problems such as deadlocks or unreceived messages on the other hand. In the context of unordered asynchronous communication, we already studied the reasons which may lead to uncontrollable service models [18]. Further impacts of synchronous communication on controllability are reported by Wolf [23] and Wolf [22].

4 Conclusion

We briefly studied communication models. By sketching different dimensions, we refined and systematized the vague synchronous/asynchronous distinction which can be found in the literature. Furthermore, we demonstrated the impact of different communication models with respect to controllability. We showed that one and the same service model (given as BPMN diagram) can be controllable or uncontrollable depending on the chosen communication model.

Future Work

Directions of future work are manifold. First, we plan to extend the dimensions of the communication models with other aspects related to instantiation semantics [10], lossy channels, or topologies. Second, this more detailed distinction can be used to classify and categorize further service formalisms from literature with respect to their assumed communication model. This would not only help to better compare and transfer results, but also to understand which communication model is required to study certain properties. We already sketched the impact to controllability in Sect. 3. Similarly, Kazhamiakin and Pistore [15] study the impact of communication models to choreography realization [12] and provide an algorithm which finds the "simplest" communication model under which a given choreography can be realized.

There already exist several approaches to translate between different communication models. Fu et al. [13] investigate necessary conditions for *synchronizability*, viz. when it possible to safely abstract from channels. This abstraction is motivated by the availability of more efficient verification techniques for synchronously communicating services. The converse direction from synchronous to asynchronous communication, called *desynchronizability*, is studied by Decker et al. [9]. They report of problems that may arise if a synchronous service choreography is implemented by asynchronously communicating services. Here, the transformation is motivated by the statement that atomic synchronous communication is an unrealistic assumption in the area of inter-organizational business processes.

Finally, the classification further needs to be further differentiated against many approaches to characterize several aspects in the area of business processes and services in terms of*patterns*. Whereas the control flow of a service can be investigated using *workflow patterns* [1], other approaches such as *enterprise integration patterns* [14], *service interaction patterns* [4], or *PAIS patterns* [20] also take interaction into account. Consequently, we plan to investigate similarities between interaction models and these patterns.

References

1. Aalst, W.M.P.v.d., Hofstede, A.H.M.t., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
2. Alur, R., Etessami, K., Yannakakis, M.: Inference of message sequence charts. IEEE Trans. Software Eng. 29(7), 623–633 (2003)

3. Alves, A., et al.: Web Services Business Process Execution Language Version 2.0. OASIS Standard, OASIS (2007)
4. Barros, A.P., Dumas, M., Hofstede, A.H.M.t.: Service interaction patterns. In: BPM 2005. pp. 302–318. LNCS 3649, Springer (2005)
5. Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Mecella, M.: Automatic composition of e-services that export their behavior. In: ICSOC 2003. pp. 43–58. LNCS 2910, Springer (2003)
6. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM 30(2), 323–342 (1983)
7. Breugel, F.v., Koshkina, M.: Models and verification of BPEL (2006), unpublished manuscript, available at `http://www.cse.yorku.ca/~franck/research/drafts/tutorial.pdf`
8. de Alfaro, L., Henzinger, T.A.: Interface automata. In: ESEC 2001. pp. 109–120. ACM (2001)
9. Decker, G., Barros, A., Kraft, F.M., Lohmann, N.: Non-desynchronizable service choreographies. In: ICSOC 2008. pp. 331–346. LNCS 5364, Springer (2008)
10. Decker, G., Mendling, J.: Instantiation semantics for process models. In: BPM 2008. pp. 164–179. LNCS 5240 (2008)
11. Desel, J.: Controlling Petri net process models. In: WS-FM 2007. pp. 17–30. LNCS 4937, Springer (2007)
12. Fu, X., Bultan, T., Su, J.: Conversation protocols: a formalism for specification and verification of reactive electronic services. Theor. Comput. Sci. 328(1-2), 19–37 (2004)
13. Fu, X., Bultan, T., Su, J.: Synchronizability of conversations among Web services. IEEE Trans. Software Eng. 31(12), 1042–1055 (2005)
14. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Professional (2003)
15. Kazhamiakin, R., Pistore, M.: Analysis of realizability conditions for Web service choreographies. In: FORTE 2006. pp. 61–76. LNCS 4229, Springer (2006)
16. Kazhamiakin, R., Pistore, M., Santuari, L.: Analysis of communication models in web service compositions. In: WWW 2006. pp. 267–276. ACM (2006)
17. König, R., Quäck, L.: Petri-Netze in der Steuerungstechnik. Verlag Technik, Berlin (1988)
18. Lohmann, N.: Why does my service have no partners? In: WS-FM 2008. pp. 191–206. LNCS 5387, Springer (2009)
19. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann (1996)
20. Mulyar, N.: Patterns for process-aware information systems: an approach based on colored Petri nets. Ph.D. thesis, Technische Universiteit Eindhoven, Eindhoven, The Netherlands (2008)
21. OMG: Business Process Model and Notation, V1.1. OMG Available Specification, Object Management Group (2008)
22. Wolf, K.: Does my service have partners? LNCS T. Petri Nets and Other Models of Concurrency 5460(2), 152–171 (2009)
23. Wolf, M.: Synchrone und asynchrone Kommunikation in offenen Workflownetzen. Studienarbeit, Humboldt-Universität zu Berlin, Berlin, Germany (2007), (in German)