Formalizing Service Interactions

Gero Decker, 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
{decker,puhlmann,weske}@hpi.uni-potsdam.de

Abstract. Cross-organizational business processes are gaining increased attention these days, especially with the service oriented architecture (SOA) as a realization for business process management (BPM). In SOA, interaction agreements between business partners are defined as choreographies containing common interaction patterns. However, complex interactions are difficult to specify, basically because a formal, common standard supporting all interaction patterns is missing. This paper motivates the use of the π-calculus for formally representing service interaction patterns.

1 Introduction

Service-oriented architectures (SOA) as a realization for business process management (BPM) aim at closely supporting business processes within a company and between business partners [1,2]. Services are employed to perform tasks within these processes and processes themselves can be exposed as services. It is distinguished between orchestrations where one business partner enacts a set of services in a given order and choreographies which represent the interaction protocols between several business partners [3]. In a setting where the different business partners encapsulate their business logic as services, service interactions are at the center of attention. A lot of effort has been undertaken to identify the most common interaction scenarios from a business perspective, which have been published as *Service Interaction Patterns* by Barros et al.[4]. Barros et al. categorize the patterns according to the number of participants in an interaction (bilateral vs. multi-lateral), the maximum number of exchanges (single-transmission vs. multi-transmission interactions) and whether the receiver of a response is necessarily the same as the sender of a request (round-trip vs. routed interactions).

The service interaction patterns are only described textually, together with business examples and design choices. The authors also come up with implementation examples using BPEL and other standards from the WS-* stack. However, the textual descriptions do not allow choreographies to be modeled else than by using textual descriptions again. The BPEL examples lack support for different service interaction patterns, thus leaving the modeler with only a subset

of possibilities. Furthermore, both kinds of descriptions lack support for formal reasoning on interaction properties like conformance, reliability, or deadlock freedom.

To overcome the limitations of expressiveness in existing notations and to allow formal reasoning, we propose formal representations of service interaction patterns. When looking into BPM literature, Petri nets in all their different flavors dominate the research community. However, Petri nets lack the ability of easily representing *mobility*, a key feature for describing dynamic structures as required in SOA. Instead, we propose the use of a process algebra, π-calculus, for formalizing service interaction patterns. Interaction and mobility form the core aspects of π-calculus and are also at the heart of the service interaction patterns.

The remainder of this paper is organized as follows. It starts by investigating related work. This is followed by discussing a subset of interesting interaction patterns in the π-calculus. Finally, a conclusion is drawn and an outlook is given.

2 Related Work

Recently several papers have been published that deal with formalizing web service choreographies, e.g. [5,6], or Busi et al. [7]. All these approaches are based on process algebras other than π-calculus. Busi et al. argue that mobility, a key feature of the π-calculus, is not needed for describing service choreographies. They assume that all interaction participants are known at design-time. Petri net based approaches from Martens [8] or van der Aalst et al. [9] make the same assumptions. Moreover, Petri nets already fail in representing all workflow patterns [10], leading to the development of a new orchestration language called YAWL [11]. However, all these publications and standards like WS-CDL consider only one-way- and simple request-response-interactions. This is heavily criticized by Barros et al in [3]. Puhlmann and Weske have formalized all the workflow patterns [10] using the π-calculus [12]. This allows for translating service orchestrations into π-processes. Puhlmann et al. have already sketched in [13] how π-calculus can be used for formalizing service invocations and represent correlations. There has not been a formalization of the service interaction patterns so far.

3 Formalizing Interaction Patterns using Pi-Calculus

At the center of π-calculus are processes that interact with each other. The communication channels as well as the messages sent over these channels are called names. Channels can be passed as messages to other processes and be used for interaction later on. This capability is called link passing mobility. It allows smart solutions for formalizing the service interaction patterns. The following subsections introduce how.

3.1 The Pi-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]. The syntax of the π-calculus processes is given by:

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

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, and $!P$ is an infinite number of copies of P. $\mathbf{0}$ is inaction, a process that can do nothing, $M + M'$ is the exclusive choice between M and M'. 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, thus 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 equals y. We utilize upper case letters for process identifiers and lower case letters for names. The abbreviation $\sum_1^m(M)$ is used to denote the summation of m choices, $\prod_1^m(P)$ denotes the composition of m parallel copies of P, and $\{\pi\}_1^m$ denotes m subsequent executions of π. Furthermore defined processes are used for parametric recursion, that is $A(y_1, ..., y_n)$.

3.2 Interactions in the Pi-Calculus

In the pattern representations each interaction participant is modeled as a π-calculus process. In the case of bilateral interactions we named them A and B, in the case of multi-lateral interactions A, B_i and P where $i = 1, 2, \cdots$. Since timers and exception handling are explicitly called for in the patterns, we introduce an environmental process $\mathcal{E}_X$ per interaction participant ($X = A, B, B_i, P$). It is left open how timeouts and exception handling are implemented. $\overline{settimer_{\mathcal{E}_X}}\langle timer\rangle$ is supposed to set a new timer where a timeout is thrown by sending on channel $timer$. Exceptions can be thrown by sending on channel $\overline{fault_{\mathcal{E}_X}}$.

In the π-calculus a message represented by a name is synchronously sent and received, resulting in an interaction. I.e. if a process wants to send a message then it blocks until a receiver actually receives the message. Therefore, the π-calculus assumes synchronous communication as well as reliable and guaranteed delivery as the default case. The following subsections present formalizations for selected service interaction pattern. We omit the termination symbol $\mathbf{0}$ in process definitions for simplicity. The pattern descriptions can be found at [4].

3.3 Single-transmission Bilateral Interaction Patterns

Send: A party sends a message to another party. The pattern definition distinguishes between blocking send and non-blocking send. In the case of blocking send the sending process cannot proceed until it can be sure that the message has been received. As already mentioned above this blocking behavior is inherent to π-calculus. Blocking send is given by:

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

This pattern formalization leaves it open if the receiver of the message is known at design-time or not. If the system is defined as

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

then A knows the link to B at design-time. If it is defined as

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

then A would get the link to B at run-time. In this case D could be something like a UDDI directory where the receiver can be looked up. A' and B' represent the so called continuations mentioned in the pattern descriptions. We continue with non-blocking send:

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

Strictly speaking, the formalization for B could be omitted. However, for illustration purposes one possible implementation for B is provided to have a valid choreography. Most interaction patterns describe the interactions from the perspective of one single participant. In order to get a minimal choreography, several patterns have to be plugged together (e.g. send for A and receive for B).

3.4 Single-transmission Multilateral Interaction Patterns

Racing incoming messages: A party expects to receive one among a set of messages. These messages may be structurally different (i.e. different types) and may come from different categories of partners. The way a message is processed depends on its type and/or the category of partner from which it comes. Normally names are not typed in π-calculus. In order to retrieve the type of a message, a second name representing the type could be used. We opted for a more elegant way: for each type a channel is created and thus the channel a message is sent over determines the message's type. In the following formalization it is assumed that there are two different types of messages. Each B_i can send messages over channel a_1 if it is of the first type or over channel a_2 for the second type. Depending on the type of the message the continuation for A is either A'_1 or A'_2. The

pattern distinguishes between discarding remaining messages and keeping them for further interactions. If remaining messages are not discarded, the patterns is defined by:

$$A = (a_1(msg).A'_1 + a_2(msg).A'_2)$$
$$B_i = (\overline{a_1}\,\langle msg\rangle\,.B'_i + \overline{a_2}\,\langle msg\rangle\,.B'_i)\ .$$

Once again the formalization for B_i is just an example. In this case every B_i can sent messages of every type. If it should be modeled that the continuation of A depends on the category of the sender, we could define $B_i = \overline{a_1}\,\langle msg\rangle\,.B'_i$ and introduce another category $C_i = \overline{a_2}\,\langle msg\rangle\,.C'_i$. A generic formalization for an arbitrary number of different types/categories would be $A = \sum_{i=1}^{n} a_i(msg).A'_i$

3.5 Routing Patterns

Request with referral: Party A sends a request to party B indicating that any follow-up response should be sent to a number of other parties $(P1, P2, \cdots, Pn)$ *depending on the evaluation of certain conditions. While faults are sent by default to these parties, they could alternatively be sent to another nominated party (which may be party A).* While the pattern descriptions talks about a number of parties P_i, the following formalization only presents the case of one party P for better readability:

$$A = (\mathbf{v}\ a)\overline{b}\,\langle a, p, req\rangle\,.a(resp).A'$$
$$B = (\mathbf{v}\ msg)b(a, x, req).\tau_B.\overline{x}\,\langle a, msg\rangle\,.B'$$
$$P = p(a, msg).\tau_P.\overline{a}\,\langle resp\rangle\,.P'\ .$$

4 Conclusion and Outlook

In this paper we have shown how a selected subset of the service interaction patterns can be formalized. We investigated new directions based on mobile process algebra represented by the π-calculus. The concept of mobility is required if the receiver of a message is not known at runtime. Ten out of thirteen interaction patterns incorporate sending messages, where the receiver might not be known at design-time. In an extended research, we were able to express all service interaction patterns in π-calculus processes. Therefore, our final conclusion is that π-calculus is well suited for expressing the service interaction patterns. The full range of pattern formalizations as well as a direct comparison to Petri nets can be found at `http://pi-workflow.org`.

The formalizations presented in this paper can be the starting point for further work on a complete formal grounding of the intersection of the domains service oriented architectures and business process management using π-calculus. The very next step would be to show how the formalizations of the service interaction patterns can be integrated with the formalizations of the workflow patterns provided in [12]. Once we have both a choreography and corresponding

orchestrations available as π-calculus processes we can proceed with introducing conformance checking, e.g. verifying if the behavior of individual orchestrations complies to the choreography. Another area of interest is the investigation of soundness criteria for choreographies.

References

1. IBM: Web Services Architecture Overview (2000) `http://www-128.ibm.com/developerworks/webservices/library/w-ovr/`.
2. van der Aalst, W.M.P., ter Hofstede, A.H., Weske, M.: Business Process Management: A Survey. In van der Aalst, W.M.P., ter Hofstede, A.H., Weske, M., eds.: Proceedings of the 1st International Conference on Business Process Management, volume 2678 of LNCS, Berlin, Springer-Verlag (2003) 1–12
3. Barros, A., Dumas, M., Oaks, P.: A Critical overview of the Web Services Choreography Description Language (WS-CDL). BPTrends Newsletter **3(3)** (2005)
4. Barros, A.P., Dumas, M., ter Hofstede, A.H.M.: Service Interaction Patterns. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F., eds.: Business Process Management. Volume 3649. (2005) 302–318
5. Brogi, A., Canal, C., Pimentel, E., Vallecillo, A.: Formalizing Web Service Choreographies. In: Proceedings of First International Workshop on Web Services and Formal Methods, Elsevier (2004)
6. Gorrieri, R., Guidi, C., Lucchi, R.: Reasoning About Interaction Patterns in Choreography. In: M. Bravetti et al. (Eds.): Second International Workshop on Web Services and Formal Methods, LNCS 3670, Springer Verlag (2005) 333–348
7. Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., Zavattaro, G.: Choreography and Orchestration: A Synergic Approach for System Design. In: B. Benatallah, F. Casati, and P. Traverso (Eds.): ICSOC 2005, LNCS 3826, Springer Verlag (2005) 228–240
8. 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)
9. van der Aalst, W.M.P., Weske, M.: The P2P Approach to Interorganizational Workflow. In Dittrich, K., Geppert, A., Norrie, M., eds.: Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01), volume 2068 of LNCS, Berlin, Springer-Verlag (2001) 140–156
10. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow Patterns. Distributed and Parallel Databases **14(3)** (2003) 5–51
11. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language (Revised version. Technical Report FIT-TR-2003-04, Queensland University of Technology, Brisbane (2003)
12. Puhlmann, F., Weske, M.: Using the π-Calculus for Formalizing Workflow Patterns. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F., eds.: Business Process Management. Volume 3649. (2005) 153–168
13. 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)
14. Sangiorgi, D., Walker, D.: The π-calculus: A Theory of Mobile Processes. Cambridge University Press, Cambridge (2003)