An Introduction to Service Choreographies

Gero Decker[1], Oliver Kopp[2], Alistair Barros[3]

[1]Hasso-Plattner-Institute for IT-Systems, Germany
[2]Institute of Architecture of Application Systems, Germany
[3]SAP Research Centre Brisbane, Australia

Servicechoreographien – eine Einführung

An Introduction to Service Choreographies

M. Sc. Gero Decker: Hasso-Plattner-Institute for IT-Systems Engineering, Prof.-Dr.-Helmert-Str., 2-3, 14482 Potsdam, Germany
Tel: +49-331-5509-162, Fax: +49-331-5509-189, E-Mail: gero.decker@hpi.uni-potsdam.de
Gero Decker is a Research Assistant and PhD student at Hasso-Plattner-Institute (HPI) in Potsdam, Germany. His areas of research include inter-organizational business process management and service choreographies in particular. He is involved in SAP's activities in that field as affiliate researcher. He holds a MSc in software engineering from HPI.

Dipl-Inf. Oliver Kopp: Institute of Architecture of Application Systems, Universitätsstrasse, 38, 70569 Stuttgart, Germany
Tel: +49-711-7816-483, Fax: +49-711-7816-472, E-Mail: oliver.kopp@iaas.uni-stuttgart.de
Oliver Kopp is a Research Assistant and PhD student at the University of Stuttgart and focuses on distributed transactions and global fault handling in service choreographies. He holds a diploma in software engineering from the University of Stuttgart.

Dr. Alistair Barros: SAP Research Centre Brisbane, Mary St, 133, 4001 Brisbane, Australia
Tel: +61-7-32599554, Fax: +61-7-32599599, E-Mail: alistair.barros@sap.com
Alistair Barros is a research leader at SAP Research with more than 20 years of industry and applied research experience in service-oriented systems and business process management. He holds a PhD in computer science from the University of Queensland, Australia.

Keywords: Choreography Modeling; BPEL; WS-CDL; Let's Dance; H.4.1 [Office Automation]; K.1 [THE COMPUTER INDUSTRY]

Schlagworte: Choreographiemodellierung; BPEL; WS-CDL; Let's Dance; H.4.1 [Office Automation]; K.1 [THE COMPUTER INDUSTRY]

MS-ID: Heft: 50/2 (2008)	oliver.kopp@iaas.uni-stuttgart.de	March 15, 2010

Abstract

Service oriented architecture (SOA) is an architectural style for building software systems based on services. Especially in those scenarios where services implement business processes, complex conversations between the services occur. Service choreographies are a means to capture all interaction obligations and constraints from a global perspective. This article introduces choreographies as an important artifact for SOA, compares them to service orchestrations and surveys existing languages for modeling them.

Zusammenfassung

Die Service-orientierte Architektur (SOA) bezeichnet einen Architekturstil für die Entwicklung von Softwaresystemen, die auf Diensten basieren. Besonders in Szenarien, in denen Geschäftsprozesse als Dienste implementiert werden, entstehen komplexe Konversationen zwischen den verschiedenen Diensten. Servicechoreographien bieten eine globale Sichtweise auf diese Dienste, in der alle Interaktionseinschränkungen und -verpflichtungen abgebildet sind. Dieser Artikel stellt Choreographien als wichtiges Artefakt im SOA-Umfeld ein, grenzt sie ab gegenüber Orchestrierungen und gibt einen Überblick über existierende Modellierungssprachen.

1 Introduction

Building complex systems using services has been a significant trend in recent years. Services are loosely coupled components described in a uniform way that can be discovered and composed. Within this context a service-oriented architecture (SOA) is an architectural style including service providers, brokers and consumers. Seen from a business perspective it is a set of services that the business wants to expose to their customers and partners or other portions of the organization.

Several standards have emerged for implementing a SOA using web services: SOAP is available as message exchange format and WSDL as description language. However, these standards only focus on simple conversations between services, mainly one-way and request-response settings. Orchestration languages are used for implementing more complex services as a flow of service invocations. If all activities of a business process are implemented as services, an orchestration can implement the process by expressing the order of and the data flow between the individual service invocations.

The main benefit of using an orchestration layer and therefore making processes explicit is the possibility to rapidly adapt the overall system to new business requirements. Instead of hard-coding business processes, the most important decisions, i.e., the selection of services and the control flow between them, are made only in the orchestration layer while leaving the underlying services unchanged.

Services also play a major role in business-to-business (B2B) collaboration scenarios. In these settings each partner has full control over and responsibility for the execution of their own business processes. Provided that every partner implements their processes as orchestrations, complex conversations between these orchestrations occur. Service choreographies capture these conversations from a global perspective, i.e., internal service invocations within one partner are hidden. Choreographies act as specification and are often the starting point for implementing new orchestrations or for adapting existing ones.

This article surveys on service choreographies and is structured as follows: The next section will further elaborate on the lifecycle of choreographies and orchestrations. Section 3 compares different languages for choreography modeling, before section 4 concludes.

2 Choreographies and Orchestrations

A certain lifecycle can be observed for business processes and their implementation as orchestrations. We distinguish between design / verification, deployment, execution / monitoring and evaluation. During the design phase, the process model is created including control flow and data flow between the different activities. Simulation techniques can be applied for validating whether the process model reflects the desired behavior. Verification ensures that the model does not contain anomalies like deadlocks or unreachable activities. Also the detailed technical configuration, such as the selection of concrete message formats and services to be invoked, needs to be carried out. The finalized orchestration is deployed into an execution engine and concrete service endpoints and security configurations are defined. At runtime of the processes, the actual invocations of services takes place and monitoring functionality is used for observing and measuring it. During the evaluation phase the process execution history is examined and requirements for process changes are gathered which again feed into a new design phase.

As shown in Figure 1 a similar lifecycle applies to service choreographies. Different business partners or service providers come up with the choreography model reflecting the interaction obligations and constraints for their future collaboration. In other settings, choreographies are used as means for standardization. Efforts like RosettaNet yield at first creating a common vocabulary for a certain domain, in the case of RosettaNet the supply chain domain, and the identification of typical business document exchanges. Then the behavioral dependencies between these exchanges are captured, concrete message formats are chosen and other technical configurations are integrated. Like it is the case for orchestrations, choreographies also need to be simulated and thoroughly validated and verified.

During the distribution phase, the specifications for the individual services are derived from the choreography. This is where the choreography lifecycle produces an important artifact influencing the individual orchestrations: the participant behavior descriptions. These descriptions need to be adhered to by the different service providers. They summarize all interaction obligations and constraints from the perspective of the individual providers. In case the provider does not yet have a service in place, the behavior description can be used as skeleton for further implementation. In case the provider already has a service in place it might need to be adapted to satisfy the new requirements. An important technique for this phase is conformance checking: Is a process implementa-

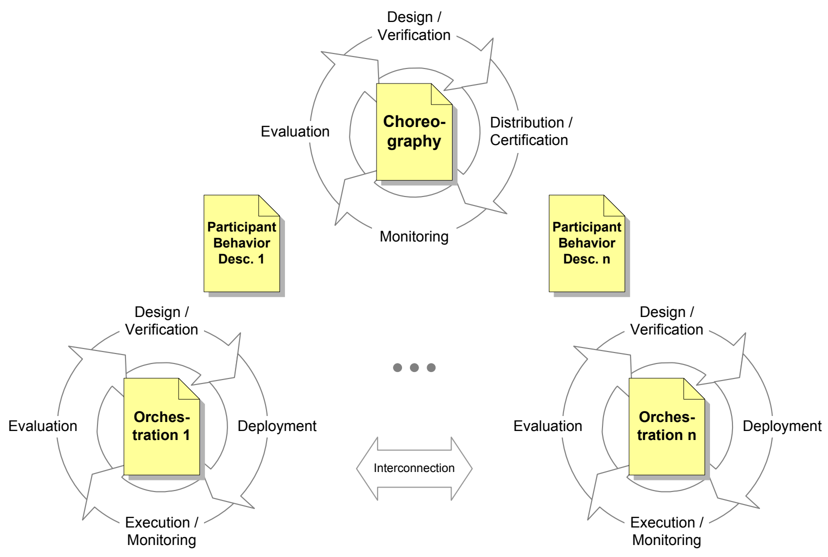

Figure 1: Lifecycle of choreographies and orchestrations

tion valid for a given specification? Typical theoretical foundations for this are weak and branching bisimulation. An interesting question is who actually carries out the conformance checks. The implementing provider could use this as self-check but also a trusted third party could be involved for certification.

Choreographies are not executed by themselves. The actual collaboration at runtime is carried out by the interconnected services, typically implemented in turn as orchestrations. However, choreographies can be used for monitoring purposes. Such an observation might serve legal purposes: In case one of the participants does not behave as specified, penalties might apply and the others might exclude him from the collaboration and chose an alternative service. Finally, an evaluation of the choreography takes place based on running or past collaboration. It might turn out that certain options in the choreography are never chosen or that certain time constraints are frequently not met. The evaluation phase can be used to optimize the overall collaboration, open it to a wider set of providers or a more general collaboration context.

3 Choreography Languages

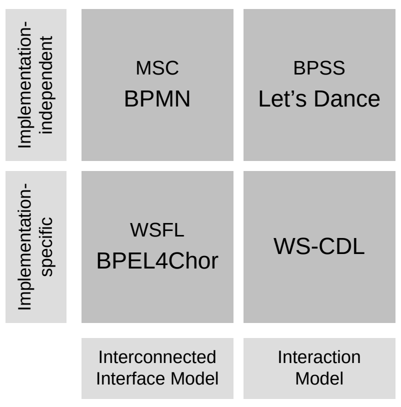

Figure 2: Categorizations of choreography languages

An important criterion for distinguishing choreography languages is the target user group. In the Business Process Management area we can generally see a difference between those seeing process models primarily as a means to describe and communicate about their business and to optimize it from a business perspective, e.g., in terms of process performance, distribution of responsibility for decisions, increase of the service quality, etc. On the other hand we find those concerned with the actual execution of business processes in IT. Both communities need to be provided with languages that best suits their needs. For example, we find e.g. the Business Process Modeling Notation (BPMN) for describing orchestrations on an implementation-independent and BPEL on a web-service-execution level.

On an implementation-independent level fundamental decisions about interactions are made. E.g., it is agreed upon whether the amount of goods delivered during the replenishment of a warehouse needs is to be ordered by the buying company or is determined by the supplier. On an implementation-specific level the concrete message formats and communication protocols as well as security issues need to be tackled. Here, it is decided whether synchronous message exchanges or asynchronous ones are to be used for certain interactions. A special challenge is exception handling, while on an implementation-independent level rather best-case modeling applies. Most choreography languages on an implementation-specific level are linked to web services.

Orthogonal to the distinction between implementation-independent and -specific levels, there are two different modeling approaches for choreographies: interaction models and interconnected interface behavior models. In case of *interaction models* elementary interactions, i.e., one-way and request-response message exchanges, are the basic building blocks. Behavioral dependencies are specified between these interactions and combinations of interactions are grouped into complex interactions. Due to the fact that these models capture the dependencies from a truly global perspective, the modeler is able to define dependencies that cannot be enforced. E.g., she might specify that a shipper can only send the delivery details to a buyer after the supplier has notified the insurance about the delivery. In this case it is left unexplained how the shipper can learn about whether the notification has been sent. Additional synchronization messages would be necessary to turn such a *locally unenforceable* interaction model into an enforceable one. In the case of *interconnected interface behavior models* control flow is defined per participant, i.e., the individual interface behavior models are stitched together using message links. Thus, such unenforceability issues cannot arise since control flow is defined per participant. However, on the other hand, interface behavior models might be *incompatible*, i.e., the different participant cannot interact successfully with each other. Deadlocks are typical outcomes of such incompatibility. A participant expecting a notification of another participant before being able to proceed and the other participant never sends such a notification.

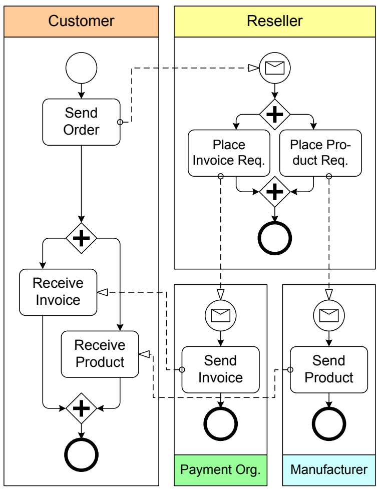

Figure 3: BPMN choreography

The Business Process Modeling Notation (BPMN [7]) is the de-facto standard for process modeling on the implementation-independent level. Since its first releases in 2004 there was a major uptake of this language and it now enjoys wide adoption by both industry and academia. The language incorporates the notion of swimlanes (called pools in BPMN), therefore allowing to assign different activities of the same process to different organizational units. An important feature for modeling choreographies is the explicit distinction between control flow and message flow. While all activities connected through control flow belong to the same process, message flow is used to interconnect different processes. BPMN comes with a rich set of constructs for expressing advanced control flow scenarios. Data flow can also be expressed through data objects associated to activities and message flow. A weak side of BPMN regarding choreographies is that every participant has to be represented by a pool. This is problematic as it is a recurrent scenario in choreographies that there are several participants of the same type involved. E.g., imagine a bidding scenario where different bidders take part in one auction. Figure 3 shows a simple choreography expressed in BPMN.

Message Sequence Charts (MSCs [4]) can also be used for describing choreographies. However, they are rather suited for describing mere sequences of interactions in contrast to full choreographies: conditional branching, parallel branching and iterations are not supported.

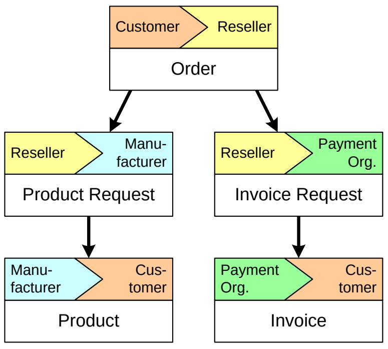

Figure 4: Let's Dance choreography

Let's Dance [8] is a language specifically designed for choreography modeling. In contrast to BPMN, Let's Dance is an interaction modeling language, i.e., atomic message exchanges are the basic building blocks of the models. Like BPMN, it targets business analysts and mainly serves to capture requirements instead of going down to implementation details. For interaction models it is not as straight forward to derive the individual interface behavior models like in the case of e.g. BPMN. Graph reduction techniques need to be applied for this purpose. Let's Dance supports more choreography scenarios than BPMN, making it a powerful yet mainly academic alternative for choreography modeling at an implementation-independent level. Figure 4 shows a Let's Dance choreography.

The ebXML initiative has come up with the Business Process Schema Specification (BPSS [2]), an implementation-independent XML-based choreography language. As a main drawback, BPSS is limited to bi-lateral choreographies only.

The Web Services Flow Language (WSFL [6]) can be used for describing choreographies of web services. Message receipt and invocation actions are defined in the different local views and then wired together in a global model. Concrete definitions of port types and operations are possible, therefore providing information on a web-services-specific level. However, the global model links operations instead of activities and thus provides a more coarse-grained view than directly linking activities.

A more recent proposal for web service choreographies is BPEL4Chor [3]. In this approach abstract BPEL processes are used for the individual participant behavior descriptions which are wired together using message links in a so called participant topology. Another special feature of BPEL4Chor is the separate definition of participant groundings. That means, the behavior descriptions and the topology do not contain technical configuration information. Therefore, it is easy to reuse a choreography for different technical setups, e.g., with different port types used. Like Let's Dance, BPEL4Chor supports all common choreography patterns from [1].

The Web Services Choreography Description Language (WS-CDL [5]) is an interaction modeling languages for web services choreographies. Being a candidate recommendation by the W3C since 2005, WS-CDL has been heavily criticized for being too different from the popular orchestration language BPEL. WS-CDL's control flow constructs differ significantly from those of BPEL, making it hard to properly generate abstract BPEL processes for the individual services out of WS-CDL choreographies. Furthermore, it turned out that WS-CDL only partially supports those choreography scenarios where a number of services of the same type are involved.

Reasoning on choreographies is an important aspect in the verification and certification phases. The most typical formalisms for this purpose are Petri nets and pi-calculus. There is a variety of techniques available for these two formalisms regarding compatibility and conformance checking. In order to use these capabilities different mappings from the choreography languages presented above to their formal representation are in place.

4 Conclusion

This article has provided an overview of service choreographies and introduced the main concepts of that field. The lifecycle of choreographies and its relationship to the lifecycle of orchestrations was discussed. As main part, different choreography languages covering both the implementation-independent level and the implementation-specific level were shortly presented and categorized.

The authors are actively involved in the advancement of choreography languages and investigate their applicability as well as formal properties.

References

[1] A. Barros, M. Dumas, and A. ter Hofstede. Service Interaction Patterns. In *BPM*, Vienna, Austria, Sept 2005.

[2] J. Clark, C. Casanave, K. Kanaskie, B. Harvey, N. Smith, J. Yunker, and K. Riemer. ebXML Business Process Specification Schema Version 1.01. Technical report, UN/CEFACT and OASIS, May 2001.

[3] G. Decker, O. Kopp, F. Leymann, and M. Weske. BPEL4Chor: Extending BPEL for Modeling Choreographies. In *ICWS*, Salt Lake City, USA, July 2007.

[4] ITU-T. Message sequence chart. Recommendation Z.120, ITU-T, 2000.

[5] N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon. Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation. Technical report, November 2005. `http://www.w3.org/TR/ws-cdl-10`.

[6] F. Leymann. Web Services Flow Language (WSFL 1.0), May 2001.

[7] Business Process Modeling Notation (BPMN) Specification, Final Adopted Specification. Technical report, OMG, Feb 2006. `www.bpmn.org/`.

[8] J. M. Zaha, A. Barros, M. Dumas, and A. ter Hofstede. A Language for Service Behavior Modeling. In *CoopIS*, Montpellier, France, Nov 2006.