TOWARDS CHOREOGRAPHY-BASED PROCESS DISTRIBUTION IN THE CLOUD

Sebastian Wagner, Oliver Kopp, Frank Leymann
Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstr. 38, 70569 Stuttgart, Germany
firstname.lastname@iaas.uni-stuttgart.de

***Abstract*—Choreographies provide means to describe collaborations. Each partner runs its own processes. To reduce the amount of data exchanged and to save resources, part of the choreography can be run on a community cloud. We show how private parts of a choreography can still be run on-premise and how non-private parts can be merged to make use of the cloud infrastructure.**

***Keywords*-Process Merge, Choreography, BPEL**

I. Introduction

Cloud computing offers a variety of new possibilities to enterprises. Advantages such as elasticity, ease-of-management, and the pay-per-use cost model ([1], i. e. that payments only depend on the resource usage) provide a great potential for reducing IT costs. Besides these advantages, the community cloud model [2] offers opportunities for cross-enterprise collaborations, i. e. scenarios where different enterprises working jointly together to achieve a certain business objective (e. g., manufacturing a good). The community cloud model enables collaborating enterprises to share the same cloud infrastructure. This does not only decrease the costs (as they can be split between the parties) but also fosters more efficient communication between the enterprises since they are using the same infrastructure. Moreover, collaboration tools for inter-enterprise cooperations can be provided by the community cloud (e. g., content management services). Besides sharing the IT resources also the business processes of the collaborating enterprises have to interact with each other (e. g., to share data).

The interaction behavior between processes can be modeled via choreographies that define the control flow relations between the involved processes [3]. To distribute parts of a single choreography and their corresponding processes onto different clouds two challenges are addressed by this work: (i) To execute parts of one process on different clouds or to execute one part in the cloud while the other part has to be performed on-premise, the process has to be split into process fragments. (ii) If interacting processes of the choreography have to be executed in the same cloud the number of process instances can be decreased by merging these processes. This can significantly improve the execution efficiency of the choreography.

The remainder of this paper is structured as follows. Firstly, a scenario for the choreography-based process distribution is presented in Section II that also serves a motivation for this work. Necessary background on BPEL and BPEL4Chor is provided in Section III. Subsequently, a short overview on process decomposition is given in Section IV. As main contribution this paper presents a technique to merge two business processes that are interconnected with a BPEL4Chor choreography in Section V. Section VI presents an overview of related work in the field and Section VII concludes and presents an outlook on future work.

II. Motivating Scenario

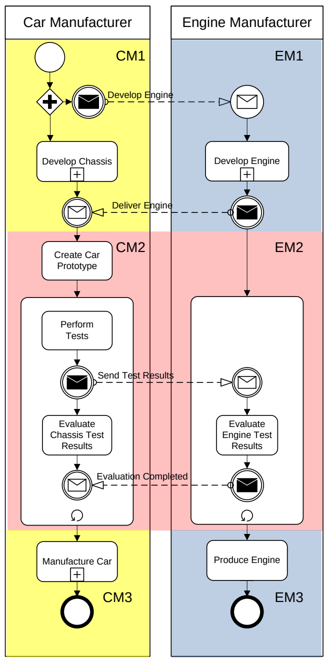

Figure 1. Example Car Development Choreography

Figure 1 depicts a choreography for creating sports cars. For the sake of simplicity, only two parties are involved in the development: the car manufacturer (briefly CM) and the engine manufacturer (briefly EM). Firstly, the CM orders an engine from the EM. Subsequently, both parties develop a prototype of the chassis and engine respectively, this is depicted by the collapsed sub process elements. After the chassis and the engine have been developed, the engine is delivered to the CM, which creates a first prototype of the sports car jointly with engineers of the EM. Then, a number of consecutive tests, depicted by the loop, are performed with the new car at the site of the CM (e. g., wind tunnel tests). The test results are shared with the EM to enable both parties to evaluate them and to perform chassis or engine modifications (not depicted here). The EM informs the CM when the evaluation of the test results was completed. Then, the next test case is performed. After all tests were completed the final version of the sports car is manufactured.

CM and EM engineers develop and test the prototype together at the CM site. To enable the engineers of both companies to access their IT infrastructure and to share the test data efficiently [4], they decide to pool those IT resources that are required for prototype development and testing in a community cloud. We assume that a workflow engine is hosted in the community cloud where all activities of the choreography dealing with prototype development and testing are executed on. This is indicated by the two process fragments in the red area, i. e. CM2 and EM2. As the CM and EM do not want to expose their development processes, these choreography parts are not performed in the community cloud, but on-premise, i. e. at the private IT infrastructure of CM (yellow area: CM1 and CM3) and EM respectively (blue area: EM1 and EM2).

The choreography model itself is rendered using a BPMN 2.0 collaboration diagram [5], but stored in BPEL4Chor as described by Decker et al. [6]. In BPEL4Chor, the CM process and the EM process are each stored as BPEL process. CM and EM are listed as participants in the BPEL4Chor topology, where the message links "DevelopEngine", "DeliverEngine", etc. are defined to connect the respective communication activities.

In Section IV, we describe how the process models can be split into the fragments CM1, CM2, CM3, and EM1, EM2, EM3 respectively. The straight-forward approach is to provision and run the processes as they are on the respective workflow engines, i. e. CM1, CM3 on the engine of the CM; EM1, EM3 on the engine of the EM; and CM2, EM2 on the engine in the community cloud. However, as two associated processes are running on each engine several performance and cost disadvantages arise. First of all two instances have to be created on each engine, resulting in six instances for the execution of the whole choreography. Only two instances were required two execute the original choreography. In this scenario this might be tolerable as it is not very likely that the choreography is executed that often because a new car will be only developed every few months or even years. However, as stated in [7], scenarios exists, for instance in the banking domain, where a large amount of process instances are performed every day. In these scenarios a large amount of instances increases the resource usage and can become a performance issue as lots of instances might throttle the underlying IT infrastructure, e. g. the workflow engine or the databases. Especially, if the infrastructure is hosted in the cloud the increasing resource utilization leads to additional costs as cloud providers usually charge on a pay-per-use basis [8]. Another negative impact on the overall performance of the choreography is the communication overhead between processes that are running on the same workflow engine (e. g. between process fragments CM1 and CM3). This overhead is caused by the SOAP messages that are exchanged between the process instances as the messages have to be usually serialized to the XML-based SOAP format at the sender's side and de-serialized at the receiver side (i. e. transformed to the internal object structure of the engine). Davis Parashar [9] and Ng et al. [10] showed that the serialization and de-serialization of complex SOAP messages can be costly. Additionally, message correlation has to be performed in order to dispatch the message to the correct process instance [11]. Again, the message serialization and correlation may not be problem if only a few instances are executed, however, in instance-intensive workflows this provides a significant overhead, even though the instances reside on the same engine and no network traffic is generated. In order to improve the performance, i. e. to reduce the number of process instances and the corresponding message exchanges between them, we propose to merge all processes that have to be deployed on the same engine. Hence, CM2, EM2 are merged to deploy them as single process in the community cloud. Also the fragments CM1, CM3 and EM1, EM3 that will run on-premise at CM and EM respectively should be merged.

III. Background

This section provides an overview of elements of BPEL and BPEL4Chor forming the basis for the splitting and consolidation described in the subsequent sections.

A choreography encompasses the relations between the process activities of the partners and the order of the interactions. We use interconnection choreography models, which show the behavior of each participant modeled as business process and where communication activities are wired together to show the message flow (cf. [12]). We focus on the Web Service Business Processes Execution Language (BPEL [11]) as BPEL is still the de-facto language to describe workflows. The principles of our work, however, are not tied to BPEL but can be applied to other process description languages such as BPMN. We use BPEL4Chor [12] as language to describe the choreography: Each process is modeled as BPEL workflow and the interconnection is described in a BPEL4Chor topology.

BPEL offers block-structured and graph-based modeling [13]. Even if BPEL requires properly nested activities, all children of the `flow` activity may be connected using control links, which in turn introduce a control flow dependencies. All activities without an incoming link are executed as soon as the `flow` activity is executed. Each activity carries a join condition which is evaluated as soon as the status of each incoming link is known. In case the join condition evaluates to true, the activity is executed. In case it evaluates to false, the status of all outgoing link is set to “false”. This control flow semantics is called “dead-path elimination” [11].

BPEL offers to model synchronous and asynchronous send/receive message exchanges [14]. A synchronous exchange is modeled using an `invoke` activity at the sender's side and a pair of `receive`/`reply` activities at the receiver's side. In this case, BPEL4Chor requires two message links to be specified: One from the `invoke` to the `receive` and one from the `reply` to the `invoke`. An asynchronous message exchange is modeled by two one-way exchanges: `invoke` and `receive` on the sender's side and `receive` and `invoke` on the receiver's side. Each `invoke`/`receive` pair is connected using a message link.

IV. Process Decomposition

To perform the test-related activities (i. e., fragment CM2 and EM2) in the community cloud and the other activities on-premise at the side of the respective partner, the two process models have to be split into independent process models. Hence, six process fragments (CM1, CM2, CM3, and EM1, EM2, EM3) have to be created.

Khalaf and Leymann [15] describe a technique to decompose BPEL process models. The splitting consists of two aspects: (i) splitting the control links and keep operational semantics and (ii) split the loops and scopes. Scopes are BPEL's way to enable fault handling and compensation handling. Control links can be split using BPEL's built-in mechanisms. BPEL's semantics for control links is based on dead path elimination, where “alive” or “dead” is propagated among the control links. This semantics is coincident with the execution semantics of BPMN 2.0 in the case of acyclic processes [16]. For modeling cycles, BPEL enforces explicit looping activities. For coordinating split loops and split scopes, additional WS-Coordination based middleware is needed [17]: Each process model registers itself at a WS-Coordinator and sends notifications to it in case of a completion of a split loop or a state change in the split scope. In turn, the coordinator informs the other processes of the completion or repetition of the loop and about changes in the state of the split scope.

V. Choreography-based Process Consolidation

We merge two processes by replacing message exchanges by control flow relationships. Each send/receive pair interconnects the two processes via a message link and thus induces an implicit control flow relation between the two processes (e. g. the CM activity Receive Engine cannot be performed until the EM activity Send Engine was executed). We transform this implicit dependency into an explicit control flow dependency. In the following, we describe the consolidation of two processes. A repeated application allows merging of an arbitrary number of processes.

The consolidation operation requires a BPEL4Chor choreography as input that encompasses two participant behavior descriptions being the process models P_i and P_j. We require that variables must be specified in the send/receive activities. This information is used to generate data-flow in Section V-B.

A. Control flow Generation

The general idea of the consolidation is to replace the communication activities with synchronization activities. We use `assign` activities for synchronization. Here, we ensure right embedding in the control flow and Section V-B describes which data is copied by the generated `assign` activity.

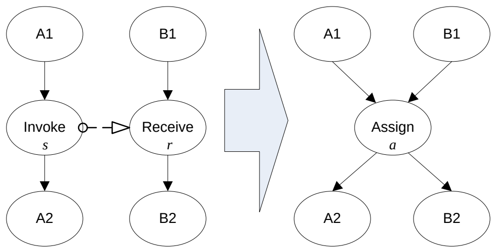

Figure 2. Asynchronous Consolidation

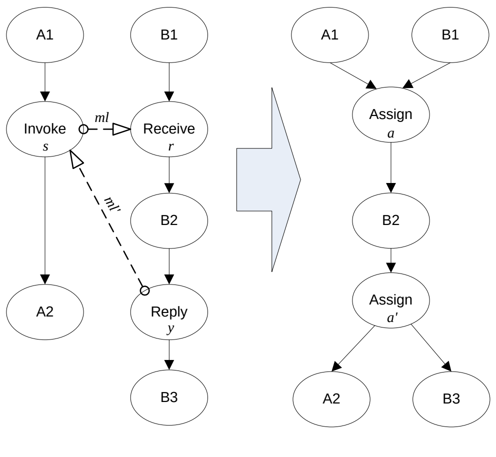

Figure 3. Synchronous Consolidation

First of all, the consolidation puts the activities, control links, and variables of the two processes into P^{new}. The activities are put into a `flow` activity, which leads to a potential parallel execution of the nested activities. During the conversion, control links are added. They ensure that the execution order of the activities is the same as in the unmerged case. In case variables of P_i and P_j carry the same name, they are renamed to ensure unique variable naming.

For each message link ml in the set of message links of the choreography, following steps are taken:

1) Let s be the sending activity and r be the receiving activity referenced by the message link ml.
2) If s is a `reply` activity, skip the link: The link models the reply to a synchronous `invoke` activity, where two message links are connected to. One where the `invoke` is the sender and one where the `invoke` is the receiver. The `invoke` is only handled once when reaching the message link, where the `invoke` is the sending activity. Thus, the other case does not need to be handled.
3) If s is an *asynchronous* `invoke` activity, remove s and r and add a new `assign` activity a. The incoming links of s and r become the incoming links of a. The join condition of a is the logical OR of the join conditions of s and r. The set of outgoing links of s and r become the set of outgoing links of a. This is illustrated in Figure 2.
4) If s is a *synchronous* `invoke` activity, there is a message link for the reply message. Here, s is the receiving activity and there is a reply activity y, which is the sending activity. A new `assign` activity a is added and s and y are removed. As depicted in Figure 3 the `reply` activity y is replaced by a new `assign` activity a'. Similar to the asynchronous case, the set of incoming links and the respective join conditions of s and r are joined to a. The outgoing links are handled differently: The `receive` activity r finishes as soon as the request message from the `invoke` activity s is received. Thus, the outgoing links of r are put into a. As the `invoke` activity s is finished as soon as the message from the `reply` activity y is received, the outgoing links of s and y are merged into a'.
5) Remove ml from the participant topology. In case of s being a synchronous `invoke` activity, remove ml', too.

B. Data Flow Generation

In the last section, the communication activities have been replaced by `assign` activities ensuring correct synchronization of the two process models merged into one process model. Besides synchronization, they are used to transform the former message flow between two activities into a data flow. Before consolidation, the message to be sent is stored in a variable. Let this variable be v_s. The message is then transmitted, received, and stored in another variable by the other process. Let this variable be v_r. To convert the message exchange to a data exchange there are two options:

(i) We copy the content of the variable v_s to the variable v_r.
(ii) We replace all occurrences of v_r *after* the receipt of the message by v_s.

```
<assign>
  <copy>
    <from variable="SndTestResultsMsg"/>
    <to variable="RcvTestResultsMsg" />
  </copy>
</assign>
```

Listing 1. Generated assign activity for hand-over of the test results

For example, assume that the message link *SendTestResults* is realized as follows in CM and EM: The variable `SndTestResultsMsg` contains the test results generated by CM. The received test results are stored by EM in the variable `RcvTestResultsMsg`. This leads to the following activities:
s: `<invoke inputVariable="SndTestResultsMsg"/>`
r: `<receive variable="RcvTestResultsMsg"/>`. In case the first approach is used, the generated `assign` activity copies `SndTestResultsMsg` to `RcvTestResultsMsg` (Listing 1). This approach is always valid as the variables v_s and v_r are kept separated and the data transfer by a message exchange is "emulated" by copying the data from v_s to v_r.

The major disadvantage of the first approach is that data is stored twice after the assign activity. This duplication can especially become a performance issue if the variables contain complex or large data elements. Rather than copying the values, approach ii) can be taken: Instead of copying the value of v_s to v_r, all accesses to v_r are replaced by accesses to v_s. The `assign` activity is replaced by an `empty` activity as no data has to be copied. This approach, however, becomes invalid if the original v_s is used after the message exchange: Formerly independent variables are now used as one, which leads to the lost update problem [18]. To solve this issue, we start with approach i). Afterwards, we determine the explicit data flow of the process model as shown by Kopp et al. [19]. In case, there is no access to v_s after the respective assign activity, we replace all accesses to v_r to accesses to v_s. Subsequently, the `assign` activity is replaced by an `empty` activity as the former `assign` activity does not copy any data, but is only used for synchronization.

VI. Related Work

In our approach interacting processes are merged: the processes are designed to work together. Most of the related work deals with the consolidation (or merging) of processes that are semantically equivalent. That means, two versions of the same process are merged. This is different from our approach as we aim to merge processes that complement each other. For instance, Mendling et al. [20] consolidate Event-driven Process Chains (EPC [21]): Semantically equivalent events and functions of two EPCs are merged.

Küster et al. [22] propose an approach to merge processes that origin from the same process based on change logs that contain information about the change operations that led from the original process to the new processes.

Sun et al. [23] present a consolidation operation for processes that are modeled as Petri-nets. There, merge points have to be defined manually. In our approach, the merge points are set by the communication activities connected by a message link.

In the area of software engineering merging of either independently developed software artifacts or different versions of the same artifact is an issue. Mens [24] provides an overview of software merging approaches.

VII. Conclusion and Future Work

We presented techniques to distribute parts of cross-partner choreographies in the cloud to perform collaborations between enterprises more efficiently. This encompassed a decomposition technique that is required to split the choreography (e. g., to host part of it on-premise) and also a new consolidation technique. This new technique can be applied to merge the processes of the choreography in order to reduce the number of process instances and the communication overhead between them if they are hosted in the same cloud. The consolidation of the processes in the choreography is conducted by generating control flow links from the message links that are defined between the processes.

The consolidation of BPEL loops, correlation sets, scopes, and the handlers (e. g. event handlers and fault handlers) that are attached to the scopes has to be investigated in our future work. We investigated scenarios with a fixed set of participants. There are also scenarios, where the number of participant instances is a priori unknown. Additionally, BPEL4Chor enables link passing mobility, where references to participants can be passed to make the participant known to other participants. Handling these two aspects of BPEL4Chor choreographies are the main focus of our future work.

References

[1] F. Leymann, "Cloud Computing: The Next Revolution in IT," in *Proc. 52th Photogrammetric Week*. Wichmann, 2009.

[2] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," *National Institute of Standards and Technology*, vol. 53, no. 6, p. 50, 2009.

[3] R. M. Dijkman and M. Dumas, "Service-Oriented Design: A Multi-Viewpoint Approach," *Int. J. Cooperative Inf. Syst.*, vol. 13, no. 4, pp. 337–368, 2004.

[4] T. Unger and S. Wagner, "Collaboration Aspects of Human Tasks," in *CEC-PAW*, 2010.

[5] Object Management Group (OMG), *Business Process Model and Notation (BPMN) Version 2.0*, 2011, 7OMG Document Number: formal/2011-01-03.

[6] G. Decker, O. Kopp, F. Leymann, K. Pfitzner, and M. Weske, "Modeling Service Choreographies using BPMN and BPEL4Chor," in *CAiSE*. Springer, 2008.

[7] K. Liu, H. Jin, J. Chen, X. Liu, D. Yuan, and Y. Yang, "A Compromised-Time-Cost Scheduling Algorithm in SwinDeW-C for Instance-Intensive Cost-Constrained Workflows on a Cloud Computing Platform," *Int. J. High Perform. Comput. Appl.*, vol. 24, pp. 445–456, November 2010.

[8] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 50–55, December 2008.

[9] D. Davis and M. P. Parashar, "Latency performance of soap implementations," in *CCGRID*. IEEE, 2002.

[10] A. Ng, S. Chen, and P. Greenfield, "An Evaluation of Contemporary Commercial SOAP Implementations," in *AWSA*, 2004.

[11] *Web Services Business Process Execution Language Version 2.0 – OASIS Standard*, OASIS, 2007.

[12] G. Decker, O. Kopp, F. Leymann, and M. Weske, "Interacting services: from specification to execution," *Data & Knowledge Engineering*, vol. 68, no. 10, pp. 946–972, April 2009.

[13] O. Kopp, D. Martin, D. Wutke, and F. Leymann, "The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages," *Enterprise Modelling and Information Systems*, vol. 4, no. 1, pp. 3–13, June 2009.

[14] A. Barros, M. Dumas, and A. H. M. T. Hofstede, "Service Interaction Patterns," in *BPM*. Springer, 2005.

[15] R. Khalaf and F. Leymann, "Role-based Decomposition of Business Processes using BPEL," in *International Conference on Web Services (ICWS)*. IEEE, 2006.

[16] H. Völzer, "A new semantics for the inclusive converging gateway in safe processes," in *BPM*, 2010.

[17] R. Khalaf and F. Leymann, "Coordination for Fragmented Loops and Scopes in a Distributed Business Process," in *BPM*. Springer, 2010.

[18] P. A. Bernstein and E. Newcomer, *Principles of Transaction Processing for the Systems Professional*, 2nd ed. Morgan Kaufmann, 2009.

[19] O. Kopp, R. Khalaf, and F. Leymann, "Deriving Explicit Data Links in WS-BPEL Processes," in *SCC*. IEEE, 2008.

[20] J. Mendling and C. Simon, "Business process design by view integration," in *BPM Workshops*. Springer, 2006.

[21] A.-W. Scheer, O. Thomas, and O. Adam, *Process Aware Information Systems: Bridging People and Software Through Process Technology*. Wiley-Interscience, 2005, ch. Process Modeling Using Event-Driven Process Chains.

[22] J. Küster, C. Gerth, A. Förster, and G. Engels, "A tool for process merging in business-driven development," in *Proceedings of the Forum at the CAiSE*, 2008.

[23] S. Sun, A. Kumar, and J. Yen, "Merging workflows: A new perspective on connecting business processes," *Decision Support Systems*, vol. 42, no. 2, pp. 844–858, 2006.

[24] T. Mens, "A state-of-the-art survey on software merging," *IEEE Trans. Softw. Eng.*, vol. 28, pp. 449–462, May 2002.