

# Towards Verification of Process Merge Patterns with Allen's Interval Algebra

Sebastian Wagner, Oliver Kopp, and Frank Leymann

Institute of Architecture of Application Systems, University of Stuttgart, Germany  
lastname@iaas.uni-stuttgart.de

**Abstract** Choreographies present how parties collaborate to achieve an agreed business objective. When companies are bought, their processes have to be in-sourced. Thereby, their part in a choreography has to be merged with the part of their acquiring business partner. Merging patterns may be applied to merge reoccurring activity combinations, such as send/receive. It has to be proven that each merge patterns keeps the relations of the original activities of the choreography. As a first step, we show by an example how the relations between activities may be expressed using the Allen calculus. We show for merging a synchronous message exchange, which relations have to be considered for validating an implementation of that merge.

## 1 Introduction

In today's business scenarios enterprises often have to collaborate to achieve an agreed business objective. This is especially true if sophisticated goods such as planes, cars, engines, etc. have to be developed. The steps that have to be performed by each company are usually defined by the respective business process model or orchestrations. To reach the overall business objective, the collaboration behavior between these different process models can be modeled by a choreography that describes the interaction behavior between the activities of the involved processes usually in form of message exchanges [13]. Choreographies may be modeled using interaction models or interconnection models [3]. In the following, we focus on interconnection models, where the publicly observable behavior of each participant in a choreography is modeled as process and where the communication activities are wired together.

As in-sourcing or back-sourcing becomes more and more common nowadays, the process models of the outsourced partner have to be reintegrated into the choreography. To accomplish that we introduced an approach to consolidate (merge) process models that are part of a choreography [16]: Pairs of sending and receiving activities are transformed to value-assignment activities. In ongoing work, we extend the approach to use *merge patterns* describing merges of structures such as while loops or a one-to-many send. Thereby, we want to show that the patterns keep the control flow dependencies between the activities. In other words, the control-flow dependencies between the activities in the merged choreography have to be the same as the dependencies between the activities in the original choreography. We plan to show that by using the Allen Calculus that is also referred to as interval algebra [1]. This paper presents a first informal mapping of a

subset of BPEL’s constructs to the Allen calculus. For one merge pattern, the properties to be considered are described.

Consequently, the remainder of this paper is structured as follows: Section 2 provides a brief overview about the choreography notation BPEL4Chor and the Allen calculus. Section 3 provides an overview on the merge approach and a rendering of the choreography using the Allen calculus. Section 4 presents the properties to be kept when applying the merge pattern for asynchronous communication. After discussing related work in Sect. 5, Sect. 6 concludes the work and provides an outlook.

## 2 Preliminaries

The consolidation approach that is described here is designed for BPEL process models [12] that are part of a BPEL4Chor [4] choreography as BPEL is still the de-facto standard for describing and enacting processes. Even if BPEL is not formalized, we use the understanding of one of its inventors to capture the relations between activities formally. If we use a formal meta model, the mapping of BPEL to a meta model still is subjective.

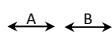
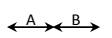
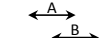
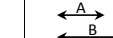


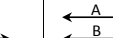
BPEL offers the `invoke` activity to send messages. In its synchronous form, it also waits until a `reply` message is received. In the asynchronous form, it solely sends a message. Messages may be received by `receive` activities. A `reply` to a synchronous call is realized by a `reply` activity. The terms “synchronous” and “asynchronous” do not state anything about the underlying messaging transport used. For instance, if Java Messaging Service [15] is used, the transport is always asynchronous even if the operation invoked at the partner is a request/response operation.

To model a choreography BPEL4Chor provides message links to interconnect the activities of the involved process models. For asynchronous `invoke/receive` communication between two processes BPEL4Chor requires that one message link has to be modeled between the two activities. In a synchronous communication scenario two message links have to be modeled, one from `invoke` to `receive` activity and another one from the `reply` to the `invoke` activity. BPEL offers a rich set of control-flow constructs. It offers block-structured constructs (such as `while` for while loops) and graph-based constructs (such as `flow` with links to model acyclic graphs) [6]. We use the graph-based part using a `flow` activity. We assume that BPEL’s dead path elimination is activated and the default join condition is used. This causes an activity to be executed if at least one of its incoming links is “not dead”.

In this paper we present the idea to use the Allen’s interval algebra to verify the correctness of a merge pattern. Currently, there is no merge pattern for BPEL’s scopes and loops. Therefore, we omit loops, event handling, fault handling, termination handling, and compensation handling in this paper.

To verify merge patterns, the control flow relations between the activities of the BPEL4Chor choreography are captured using Allen’s interval algebra [1]. This algebra defines 13 distinct basic relations that can be defined between two intervals  $a$  and  $b$  that are depicted in Fig. 1. Using these basic relations, more complex relations between two intervals can be defined, e. g., the relation  $a\{<,d\}b$  denotes that  $A$  exists either before or during  $B$ . The composition operation  $R' \otimes R''$  of two intervals  $R'$  and  $R''$  is provided to

calculate the transitive relations between the intervals  $a$  and  $c$ , where  $aR'b$  and  $bR''c$ . To derive the composition of two relations, their basic relations are pairwise composed, i. e.,  $R' \otimes R'' = \{r' \otimes r'' | r' \in R', r'' \in R''\}$ . The result of a composition of the basic relations is defined by the composition table that is provided by Allen [1]. For the intervals  $a\{<\}b$  and  $b\{<\}c$  the composition operation is  $R' \otimes R'' = <$ , i. e.,  $a$  before  $c$ .

A before B: A<B B after A: B>A 	A meets B: AmB B met-by A: AmiB 	A overlaps B: AoB B overlapped-by A: AoiB 	A starts B: AsB B started-by A: AsiB 	A finishes B: AfB A finished-by B: AfiB 	A during B: AdB B contains A: diB 	A equals B: AeB 
--	---	---	--	---	---	--

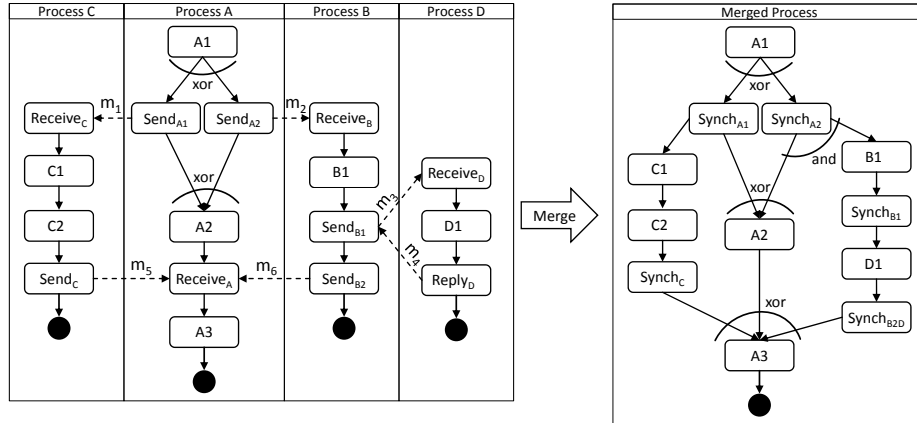
**Figure 1.** Allen's Interval Relations

In the approach described in this work we use the Allen calculus to determine the relations between activities instead of intervals. The advantage of using the Allen calculus is that it is a full algebra providing a set of operations for determining transitive relationships between activities. The graph-based part of BPEL defines predecessor and successor relationships. The block-structure of BPEL defines relations between composite activities (e. g., *while*) and their children. Allen's calculus is capable to capture both the graph-based and the block-structured part of BPEL. The *during* relation can for instance be used if we want to model the relation between a BPEL loop or a BPEL scope and its child activities. Using an equivalence notion of the linear time/branching time spectrum [5] is no option. It is not possible to express a *during* relationship as the notions treat state machines only. There are no nested states in state machines.

### 3 Choreography-based Process Consolidation

The approach of choreography-based process consolidation was introduced in [16]. Figure 2 presents an example choreography to illustrate the description using the Allen calculus. Process A sends a message to process B or process C. Process B synchronously calls a process D. A result message is sent from process B or process C to process A. The choreography has been merged into a single business process: All pairs of communication activities have been merged.

Message links in the choreography imply control flow relations between the involved processes. For instance, message link  $m_1$  implies that activities  $C1$  and  $C2$  are always performed after  $A1$  was executed. The consolidation approach replaces these implicit relations by an explicit control flow. Different interaction scenarios between the collaborating processes define different control flow relations between their activities. For instance, an asynchronous send/receive has different implications on the control flow relations between the activities of the involved processes than a synchronous send. Hence, different merge operations have to be applied. To goal is to merge process models into a single process model in a way that the explicit and implicit control flow relations specified by the choreography are kept. Consequently, the relations that exist between all activity pairs of the choreography have to be same in the new process model. Table 1



**Figure 2.** Example Choreography

depicts the pairwise relationships between the activities of the example choreography. The send and receive activities are omitted in the table as they are removed during the consolidation.

	A1	A2	A3	B1	C1	C2	D1
A1	$\emptyset$	<	<	<	<	<	<
A2	>	$\emptyset$	<	$\mathcal{R}$	$\mathcal{R}$	$\mathcal{R}$	$\mathcal{R}$
A3	>	>	$\emptyset$	>	>	>	>
B1	>	$\mathcal{R}$	<	$\emptyset$	$\mathcal{R}$	$\mathcal{R}$	<
C1	>	$\mathcal{R}$	<	$\mathcal{R}$	$\emptyset$	<	$\mathcal{R}$
C2	>	$\mathcal{R}$	<	$\mathcal{R}$	>	$\emptyset$	$\mathcal{R}$
D1	>	$\mathcal{R}$	<	>	$\mathcal{R}$	$\mathcal{R}$	$\emptyset$

**Legend:**

- Rows list the left part of relation relation
- Columns list the right part each relation
- <: set consisting of the single relation “before”
- >: set consisting of the single relation “after”
- $\emptyset$ : no relation
- $\mathcal{R}$ : all relations hold

**Table 1.** Relations between activities in the example choreography

The actual merge operation consists of several steps that are described informally in following. To simplify the description, we describe the merge of all participant behavior descriptions of a choreography into a single orchestration.

First, a new process model is created with a flow activity as top level element. All participant behavior descriptions of the choreography are copied into this flow activity to reflect the parallel execution of the orchestrations that exist in the choreography. Then, on each single interaction between two or more participant behavior descriptions a merge pattern is applied to replace the message links by control flow links. The type of pattern that is applied depends on the interaction type.

To validate the merge patterns, it has to be shown that the relation set  $R$  of the new orchestration models equals the relation set  $R_C$  of the original choreography: All atomic activities contained in the new orchestration *and* the original choreography have the

same relations. That means, the relations of all activities not removed or inserted remain the same.

## 4 Properties of Synchronous Communication

In the following section, we treat the properties a merge pattern for merging synchronous communication has to keep. In the context of BPEL, synchronous communication denotes that the activity sending the request also receives the response message. The synchronous communication pattern is implemented in BPEL4Chor by a synchronous `invoke` activity that is related to a `receive` activity via a message link  $m$ . The `receive` is directly or indirectly followed by a single logical `reply` activity that is also connected to the `invoke` via a message link  $m'$ . “Single logical `reply` activity” describes that there may be multiple `reply` activities belonging to the `receive` activity, but that there may only be one of them executed after the `invoke` has been executed. In this paper, we assume that there is exactly one `reply` activity given for a `receive` activity. Furthermore, we assume that there is exactly one `invoke` activity for the `receive` activity. BPEL4Chor allows multiple `invoke` activities for one `receive` as long as the `invoke` activities are mutually exclusive.

An example of a synchronous interaction is given in Fig. 2 where  $Send_{B1}$  is connected to  $Receive_D$  via  $m_3$  and  $Reply_D$  is connected to  $Send_{B1}$  via  $m_4$ . One important characteristic of synchronous communication is that the sender blocks until it receives the response. Technically spoken, this means that the `invoke` does not complete until it receives a message from the `reply`. This behavior has implications on the control flow relations between the activities that are depicted in Table 2.

For the proof, we require that there are no consecutive interactions between two partners. If there are, we regard that part of the process as a sequence of the first interaction, followed by an empty at each partner, followed by the second interaction. An empty activity does nothing. In short, this rewrite is necessary as we regard the direct predecessors of the communication activities and want to assume that they can happen in any order.

	$\bullet s$	$s\bullet$	$\bullet rc$	$rc\bullet^{rp}$	$rc\bullet^{\overline{rp}}$	$\bullet rp$	$rp\bullet$
$\bullet s$	$\emptyset$	$<$	$\mathcal{R}$	$<$	$<$	$<$	$<$
$s\bullet$	$>$	$\emptyset$	$>$	$>$	$\mathcal{R}$	$>$	$\mathcal{R}$
$\bullet rc$	$\mathcal{R}$	$<$	$\emptyset$	$<$	$<$	$<$	$<$
$rc\bullet^{rp}$	$>$	$<$	$>$	$\emptyset$	$\mathcal{R}$	$<$	$<$
$rc\bullet^{\overline{rp}}$	$>$	$\mathcal{R}$	$>$	$\mathcal{R}$	$\emptyset$	$\mathcal{R}$	$\mathcal{R}$
$\bullet rp$	$>$	$<$	$>$	$>$	$\mathcal{R}$	$\emptyset$	$<$
$rp\bullet$	$>$	$\mathcal{R}$	$>$	$>$	$\mathcal{R}$	$>$	$\emptyset$

**Legend:**

- $\bullet a$  – the set of all directly preceding activities of  $a$
- $a\bullet$  – the set of all directly succeeding activities of  $a$
- $s$  – the `invoke` activity
- $rc$  – the `receive` activity
- $rp$  – the `reply` activity
- $rc\bullet^{rp}$  – all direct successors of  $rc$  being on a path to  $rp$ .
- $rc\bullet^{\overline{rp}}$  – all direct successors of  $rc$  not being on a path to  $rp$ .

**Table 2.** Relations in a synchronous scenario

No statement about the relations between the direct predecessor and successor activities of  $s$  and  $rp$  can be made if just the direct predecessors and successors of  $s$ ,  $rc$ , or  $rp$  are considered. However, it is clear that dependencies must exist between  $\bullet s$  and  $\bullet rc$ , because they are transitively connected via prior control links or message links. The predecessor and successor relation within one participant behavior description (e. g.,  $rc^{\bullet rp} \{<\} s^{\bullet}$ ) are trivial as they are only defined by the control links.

Concerning the relations of the successor activities of  $rc$  two kinds of successor activities have to be distinguished, namely  $rc^{\bullet rp}$  and  $rc^{\bullet rp}$ . The successor activity  $rc^{\bullet rp}$  is no direct or indirect predecessor of  $rp$ . Hence, it has no explicit relation to the successor of the send activity, i. e.,  $rc^{\bullet rp} \{\mathcal{R}\} s^{\bullet}$ . For the successor activity  $rc^{\bullet rp}$  that resides on the path to the reply activity  $rp$  exists a direct relation  $rc^{\bullet rp} \{<\} s^{\bullet}$ . This relation is implicitly defined by the message link. As  $s^{\bullet}$  can be only performed after  $s$  completed and  $s$  can only complete after it got a response from  $rp$  which in turn is not completed before  $rc^{\bullet rp}$ . For instance, in the scenario presented in Fig. 2  $D1$  has to be performed before  $Send_{B2}$  and after  $Send_{B1}$ . This is only the case if we make the assumption that the reply activity  $rp$  completes immediately after it sent the response to the send activity  $s$ , thus  $rp \{<\} s$ . In an asynchronous communication, however, the relation  $rc^{\bullet rp} \{\mathcal{R}\} s^{\bullet}$  would exist. This is because of the operational semantics of BPEL:  $s$  sends a message to  $rc$  and completes even if  $rc$  or  $\bullet rc$  are not activated yet.

Concerning the reply activity  $rp$ , we make the assumption that it completes immediately after it sent the response to the send activity  $s$ , thus  $rp \{<\} s$ .

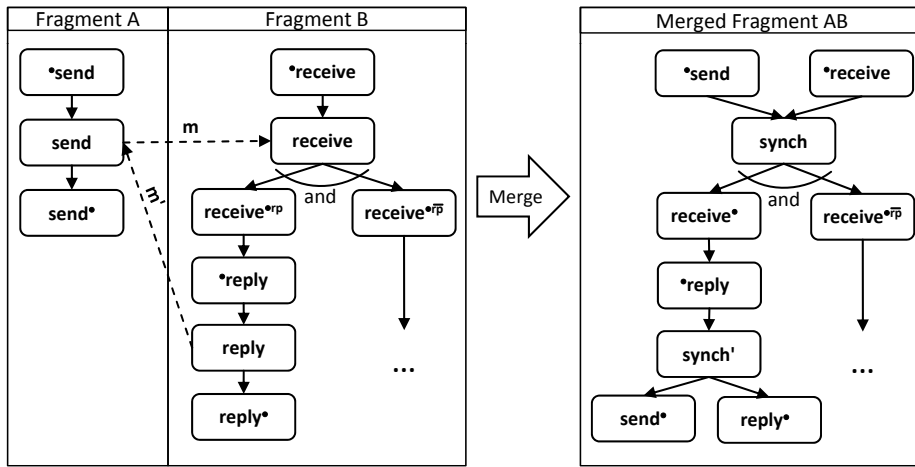


Figure 3. Merge of Synchronous Interactions

Fig. 3 depicts the merge of two process fragments that are communicating synchronously. The new orchestration at the right side of Fig. 3 keeps all control flow relations of the choreography that are sketched in Table 2. The sending activity  $s$  and the receiving activity  $rc$  are combined to a synchronization activity  $synch$ . This synchroniza-

tion activity is used to assign the values that were transported by the message  $m$  in the choreography. Likewise,  $synch'$  is inserted to assign the values that were transported in the message  $m'$  from the reply activity  $rp$  to  $s$ .

## 5 Related Work

In contrast to techniques that merge processes that are semantically equivalent we aim to merge collaborating processes. An approach for process merging is the work by Mendling and Simon [11] where semantically equivalent events and functions of Event Driven Process Chains [14] are merged. An approach to merge processes that origin from the same process using change logs is described by Küster [7].

Instead of directly generating a BPEL orchestration out of a BPEL4Chor choreography, an intermediate format may be used. There is currently no approach keeping the structure of the generated orchestration close to the structure of the original choreography. For instance, Lohmann and Kleine [9] do not generate BPEL scopes out of Petri nets, even if the formal model of Lohmann [8] generates a Petri net representation of BPEL scopes.

An overview of existing BPEL formalizations and verification approaches is provided by Breugel [2]. There is no verification approach using Allen's calculus. Lohmann et al. [10] showed how BPEL4Chor can be verified using a Petri Net representation. It is not yet shown how that mapping may be used to show equivalence between a choreography and the merged orchestration. In our work, we want to keep the ordering of the internal activities, which is more than behavioral equivalence.

Weidlich et al. [17] use behavioral profiles to capture the relations between activities in process models for compliance checking. In contrast to our approach the work does not consider the relations between activities in choreographies. Moreover, only predecessor and successor relations can be captured there. Hence, it is not possible to capture the relations between parent and child activities (block-structure) which can be accomplished with the Allen calculus.

## 6 Conclusion and Outlook

This paper presented how relations between activities may be expressed using Allen's calculus. The derivation from choreographies and orchestration has been outlined by using an example. We used the relations to show that a merge of a choreography model into an orchestration model does not change the relations of the non-merged activities.

The capturing of interval relations has been done manually. This procedure will be kept when verifying other merge patterns. This especially includes merging BPEL's scope and loop activities. To verify such patterns, we surely will have to use the *during* relation of Allen's calculus. In our future work we will investigate if we need all relations of Allen's calculus or if the subset consisting of *before*, *after*, and *during* is sufficient.

**Acknowledgments** This work was partially funded by the BMWi project Migrate! (01ME11055) and the BMWi project CloudCycle (01MD11023).

## References

1. Allen, J.F.: Maintaining Knowledge about Temporal Intervals. *Commun. ACM* 26(11), 832–843 (1983)
2. van Breugel, F., Koshkina, M.: Models and Verification of BPEL (2006), <http://www.cse.yorku.ca/~franck/research/drafts/tutorial.pdf>
3. Decker, G., Kopp, O., Barros, A.: An Introduction to Service Choreographies. *Information Technology* 50(2), 122–127 (Feb 2008)
4. Decker, G., Kopp, O., Leymann, F., Weske, M.: Interacting services: From specification to execution. *Data & Knowledge Engineering* 68(10), 946–972 (Apr 2009)
5. van Glabbeek, R.J.: The Linear Time-Branching Time Spectrum (Extended Abstract). In: *CONCUR*. pp. 278–297 (1990)
6. Kopp, O., Martin, D., Wutke, D., Leymann, F.: The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages. *Enterprise Modelling and Information Systems* 4(1), 3–13 (June 2009)
7. Küster, J., Gerth, C., Förster, A., Engels, G.: A Tool for Process Merging in Business-Driven Development. In: *Proceedings of the Forum at the CAiSE* (2008)
8. Lohmann, N.: A Feature-Complete Petri Net Semantics for WS-BPEL 2.0. In: Dumas, M., Heckel, R. (eds.) *WS-FM'07: Web Services and Formal Methods*, 4th International Workshop. *Lecture Notes in Computer Science*, vol. 4937, pp. 77–91. Springer (2007)
9. Lohmann, N., Kleine, J.: Fully-automatic Translation of Open Workflow Net Models into Simple Abstract BPEL Processes. In: *Modellierung*. *Lecture Notes in Informatics*, vol. P-127. Gesellschaft für Informatik e. V. (2008)
10. Lohmann, N., Kopp, O., Leymann, F., Reisig, W.: Analyzing BPEL4Chor: Verification and Participant Synthesis. In: *WS-FM 2007: Forth International Workshop on Web Services and Formal Methods*. Springer-Verlag (2007)
11. Mendling, J., Simon, C.: *Business Process Design by View Integration*. In: *BPM Workshops*. Springer (2006)
12. OASIS: *Web Services Business Process Execution Language Version 2.0 – OASIS Standard* (2007)
13. Peltz, C.: *Web Services Orchestration and Choreography*. *IEEE Computer* 36(10), 46–52 (2003)
14. Scheer, A.W., Thomas, O., Adam, O.: *Process Aware Information Systems: Bridging People and Software Through Process Technology*, chap. *Process Modeling Using Event-Driven Process Chains*. Wiley-Interscience (2005)
15. Sun microsystems: *JSR-000914 Java™ Message Service (JMS) API* (2002), version 1.1 April 12
16. Wagner, S., Kopp, O., Leymann, F.: Towards Choreography-based Process Distribution In The Cloud. In: *Proceedings of the 2011 IEEE International Conference on Cloud Computing and Intelligence Systems* (2011)
17. Weidlich, M., Mendling, J., Weske, M.: Efficient Consistency Measurement Based on Behavioral Profiles of Process Models. *IEEE Trans. Software Eng.* 37(3), 410–429 (2011)