Institute of Architecture of Application Systems

Enabling the Extraction and Insertion of Reusable Choreography Fragments

Andreas Weiß, Vasilios Andrikopoulos, Michael Hahn, Dimka Karastoyanova

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{andreas.weiss, vasilios.andrikopoulos, michael.hahn,
dimka.karastoyanova}@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings   {INPROC-2015-18,
    author =     {Andreas Wei{\ss} and Vasilios Andrikopoulos and Michael Hahn
                 and Dimka Karastoyanova},
    title =      {{Enabling the Extraction and Insertion of Reusable
                 Choreography Fragments}},
    booktitle =  {Proceedings of the 22nd IEEE International Conference on
                 Web Service (ICWS 2015)},
    publisher =  {IEEE Computer Society},
    pages =      {686--694},
    month =      {June},
    year =       {2015}
}
```

© 2015 IEEE Computer Society. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Universität Stuttgart
Germany

Enabling the Extraction and Insertion of Reusable Choreography Fragments

Andreas Weiß, Vasilios Andrikopoulos, Michael Hahn, and Dimka Karastoyanova
Institute of Architecture of Application Systems (IAAS)
University of Stuttgart, Stuttgart, Germany
E-mail: {*andreas.weiss, vasilios.andrikopoulos, michael.hahn, dimka.karastoyanova*}*@iaas.uni-stuttgart.de*

***Abstract*—Reuse of service orchestrations or service compositions is extensively studied in the literature of process modeling. Sub-processes, process templates, process variants, and process reference models are employed as reusable elements for these purposes. The concept of process fragments has been previously introduced in order to capture parts of a process model and store them for later reuse. However, similar efforts on facilitating the reuse of processes that cross the boundaries of organizations expressed as service choreographies are not available yet. In this paper, we introduce the concept of choreography fragments as reusable elements for service choreography modeling. Choreography fragments can be extracted from choreography models, adapted, stored, and later inserted into new models. Based on a formal model for choreography fragments, we define methods and algorithms for the extraction and insertion of fragments from and into service choreographies. We then discuss an experimental and proof-of-concept evaluation of our proposal.**

I. Introduction

Existing process modeling research has already introduced many interesting concepts and techniques enabling the reuse of executable business process models viewed as service orchestrations or compositions. Frequently used orchestration logic, for instance, can be expressed during modeling time as sub-processes to be invoked later from their "parent" process models [1], [2]. Constructs like process templates, process variants, and process reference models can also be used as reusable elements for process modeling [3]. Toward this goal, the notion of *process fragments* has been introduced in [4], [5], [6] allowing to capture parts of a process model and to store them for later (re)use. In contrast however, there is not much existing work enabling the reuse of cross-organization process models expressed as *services choreographies*.

Choreographies model the interconnection of independent participants by showing only their publicly visible communication behavior, or protocol [7]. These participants are in principle implemented as services or service orchestrations, based on their public interfaces. In this context, reuse on the level of choreographies has severe implications in decreasing the time-to-market of such process models. If, for example, a choreography modeler has the means to extract the similarities from existing choreographies such as the communication between participants, store them as reusable fragments, and insert them when modeling a new choreography spanning several systems, then there are significant gains in time spent on modeling. This observation forms the main motivation behind our work.

More specifically, in this paper we introduce the notion of *choreography fragments* in order to provide the means for reuse in choreography modeling. Choreography fragments can be used to capture best practices and recurring patterns in choreography models, store them in an appropriate library, and insert them into new choreography models. Choreography fragments can also be used for the refinement of partially defined choreography models containing abstract constructs, like the ones introduced in [8], to concrete choreography models. Similar to process fragments, choreography fragments can be created in either a top-down or bottom-up manner. Top-down entails the creation of fragments through extraction from choreography models or mining of audit trails of interconnected processes, whereas bottom-up refers to the manual creation of choreography fragments from scratch. In this work, we concentrate on the top-down approach. We base our approach on a formal model that abstracts from the actual underlying modeling language used for service choreographies and orchestrations.

The contribution of this work is therefore threefold. On the one hand, we provide a *formal model for choreography fragments* that is independent of a particular modeling language and therefore reusable across technologies. On the other hand, we provide a *method for extracting choreography fragments from choreography models, adapting, and storing them*. The necessary algorithms supporting this extraction are introduced. Furthermore, we also provide a *semi-automated process of inserting already extracted choreography fragments into choreography models* enabling their reuse.

The rest of this paper is structured as follows. We first discuss a motivating scenario (Sec. II) and provide a formal model defining the concept of choreography fragments (Sec. III). In Sec. IV, we introduce the fundamental concepts and proposed methods for extracting and inserting choreography fragments. The evaluation of the proposed concepts is then discussed in Sec. V. Sec. VI compares our work to related ones. Finally, we conclude our paper with a summary and an outlook to future work in Sec. VII.

II. Motivation

In this section, we introduce a scenario to motivate the need for choreography fragments.

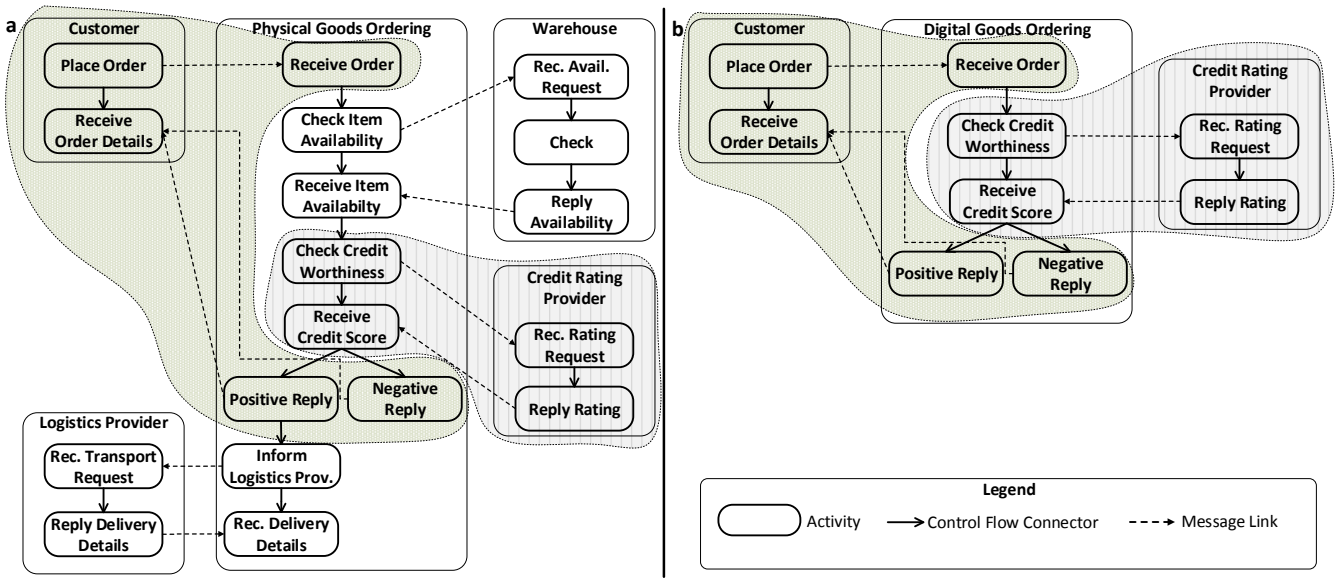


Figure 1. Motivating scenario: (a) Choreography model for ordering physical goods (b) Choreography model for ordering digital goods

Fig. 1a shows the (simplified) service orchestrations involved in the business activities of an online retail company that sells physical goods such as mobile phones or personal computers to customers. The service orchestrations form a choreography. A customer orders the goods from the web site of the retailer and triggers the ordering process. The ordering process asks the warehouse process about the item availability and invokes an external credit rating provider to assess the customers credit worthiness. Depending on the resulting credit score, the ordering process decides if the customer is allowed to order the items with the chosen payment method. In case the ordering is allowed, a positive reply is issued and a logistics provider is instructed to conduct the delivery of the physical goods.

The online retailer might decide that besides selling physical goods the distribution of digital goods would be an ideal extension of its portfolio. Its IT department, or an external consultant, now has to model and implement the relevant business processes rapidly to seize this business opportunity. The new choreography model in Fig. 1b shows that some parts of the model are similar to Fig. 1a. The interaction with the credit rating provider is still part of the choreography as well as the interaction between the customer and the ordering process. If a modeler had the means to extract these similarities from the existing choreography, store them as reusable fragments, and insert them when modeling a new choreography spanning several systems, overall time-to-market would decrease and choreography models would not have to be created from scratch. Therefore, even in small-scale choreography models like the ones discussed in Fig. 1 there exists a potential for reusability of modeling constructs that the rest of this work aims to support.

III. Formal Model

Our notion of executable process models is oriented on the work of [9], which is itself based on the formal model introduced in [1]. Our definition of process fragments adopts the formal model of [5]. Subsequently, we extend these definitions to introduce the concepts of choreography models and fragments. A choreography model consists of at least two *participants*, i.e., process models, *message links*, and explicit *control flow* and implicit *data flow* via the manipulation of *variables*. Choreography models typically only show the publicly visible communication behavior, because the details of the workflows implementing the choreography participants are considered sensitive information providing a competitive advantage to business organizations. However, the granularity of the choreography model is decided by the involved organizations and may also allow non-communication related activities and variables. The usually non-executable choreography models are defined collaboratively and used to generate abstract representations of the choreography participants in a workflow language. The collaborating business organizations then refine their workflow models with business logic.

Definition 1 formally specifies our understanding of process models, which are then used as choreography participants in a choreography model. Note that we use the projection operator π_n to access the n^{th} element of a tuple starting from index 1. $\mathcal{P}(X)$ denotes the power set of set X including the empty set $\emptyset$. $p_y.X$ accesses element X (potentially a set) of element p_y.

Definition 1 (Process Model, G)**.** A *process model* is a DAG represented by the tuple $G = (m, V, i, o, A, L)$, where $m \in M$ is the name of the process model, V is the set of *variables*, i is the *map of input variables*, o is the *map of output*

variables, A is the set of *activities*, and L is the set of *control connectors* (control flow links). Input variables providing data to activities can be assigned using an input variable map $i : A \rightarrow \mathcal{P}(V)$. Output variables to which activities may write data to are described by the output variable map $o : A \rightarrow \mathcal{P}(V)$. Finally, the set of control connectors is $L \subseteq A \times A \times C$. A link $l \in L$ is a triple $l = (a_{source}, a_{target}, t \mid a_{source}, a_{target} \in A, t \in C \wedge a_{source} \neq a_{target})$ connecting a source and a target activity, while its *transition condition* (where C is the set of all conditions) is evaluated during run time. An activity $a_{start} \in A$ is called *start activity* if it is not the target of a control flow link: $A_{start} \subseteq A := \{a \mid a \in A \wedge \forall l \in L, a \neq \pi_2(l)\}$.

Definition 2. (Process Fragment, F). A *process fragment* is a directed, acyclic graph $F \preceq_F G$ represented by a tuple $F = (m, V, i, o, A, L)$. The $\preceq_F$ operator means, that the components of the tuple F are a subset or equal to the their corresponding components in the process model tuple G. The definition of a control connector $l \in L$ is extended by the concept of *dangling control connectors*. A dangling control connector is a triple $l = (\perp, a_{target}, t)$ or $l = (a_{source}, \perp, t)$, where $\perp$ is a missing (undefined) source or target activity. Therefore, the set L of control connectors is now defined as $L \subseteq (A \cup \perp) \times (A \cup \perp) \times C$, where C is the set of transition conditions (see Definition 1).

Based on that, we define the notion of choreography model and choreography fragment. We adopt the concept of typed participants and so-called participant sets from [10]:

Definition 3. (Choreography Model, $\mathfrak{C}$). A choreography model is a directed, acyclic graph denoted by the tuple $\mathfrak{C} = (m, P, P_{set}, ML)$, where $m \in M$ is the name of the choreography model, P is the set of choreography participants, P_{set} is the set containing participant sets, ML is the set containing all message links between the choreography participants. A *choreography participant* $p \in P$ is a triple $p = (m, type, G)$, where $m \in M$ is the name of the participant, $type : P \rightarrow T$ is the function assigning a type $t_p \in T$ to the participant, and G is a process model graph as defined in Definition 1. Typing the participant allows to express if several participants of the same type participate in the same choreography. The set of all participants is denoted by P_{all}. A *participant set* $p_{set} \in P_{set}$ is described by $p_{set} \subseteq P_{all}$. This modeling construct is used to model a set of choreography participants whose number can be determined only during execution. We define a *containment relation* R_{con} as $R_{con} : P_{set} \rightarrow \mathcal{P}(P_{all})$. R_{con} indicates the participants contained in a participant set p_{set}. Contained participants in a participant set must be of the same type and have the same process model graph: $\forall p_x, p_y \in R_{con}(p_{set}) : type(p_x) = type(p_y) \wedge p_x.G = p_y.G$.

The set of *message links* ML is denoted as $ML \subseteq (P \cup P_{set}) \times P \times A \times A \times C$. A message link is a tuple $ml \in ML = (p_s, p_r, a_s, a_r, t)$, where p_s, p_r are the sending and receiving participants. Furthermore, p_s can be a participant set if any contained participant p in p_s: $p \in R_{con}(p_s)$, $p_s \in \pi_3(\mathfrak{C})$ may send something. For the sending and receiving participants the following holds: $p_s \neq p_r$, i.e., the sender and the receiver must not be identical; $a_s \in \pi_5(p_s.G)$ and $a_r \in \pi_5(p_r.G)$ are the sending and receiving activities for which the following holds: $a_s \neq a_r$. Sending and receiving activities must have only one outgoing or incoming message link. These activities are denoted as *communication activities*. The transition condition $t \in C$ is evaluated during run time. Choreography participants and participant sets are only connected via message links. However, a choreography model graph may have disconnected components because there may be participants or participant sets that are not connected to other participants or participant sets via message links. Note that the disconnection is only allowed between participants but not between activities of a participant's process graph.

Definition 4. (Choreography Fragment, $\mathfrak{F}_c$). A choreography fragment $\mathfrak{F}_c \preceq_{\mathfrak{F}_c} \mathfrak{C}$ is a directed, acyclic graph with possibly disconnected components and is represented by a tuple $\mathfrak{F}_c = (m, P_f, P_{set}, ML)$. That means that the choreography fragment tuple components are a subset or equal to the corresponding choreography model tuple components. A participant $p_f \in P_f$ is defined differently compared to the participant in a choreography model: $p_f = (m, type, F)$, where F is a process fragment. That means, in a choreography fragment a participant contains a process fragment, which can also be a process model (cf. Definition 2). The definition of a message link $ml \in ML$ is extended with the concept of dangling message links. A *dangling message link* is a tuple $ml = (\perp, p_r, \perp, a_r, t)$ or $ml = (p_s, \perp, a_s, \perp, t)$, where $\perp$ is a missing sending participant and sending activity or a missing receiving participant and receiving activity. The set of message links ML in a choreography fragment $\mathfrak{F}_c$ is defined as $ML \subseteq (P \cup P_{set} \perp) \times (P \cup \perp) \times (A \cup \perp) \times (A \cup \perp) \times C$, where C is the set of (transition) conditions. Distinct participants may be disconnected, i.e., without message links to any other participant in the choreography fragment. Depending on the selection of elements that should belong to a choreography fragment, activities inside participants may become disconnected. To represent a *valid* choreography fragment these disconnections must be repaired, either manually or automatically. The only exception are activities from parallel paths that were not connected in the original model. When creating a choreography fragment the variables an activity inside a participant reads from or writes to should also be included. Therefore, all participants from the sets of participants have to be considered: $\forall p \in \pi_2(\mathfrak{F}_c), \forall a \in \pi_5(p.F) : V_f \in \pi_2(p.F) \subseteq \{v \mid i(a) \cup o(a)\}$. The same holds for all the participants contained in participant sets: $\forall p_{set} \in \pi_3(\mathfrak{F}_c), \forall p \in R_{con}(p_{set}), \forall a \in \pi_5(p.F) :$
$V_f \in \pi_2(p.F) \subseteq \{v \mid i(a) \cup o(a)\}$.

Figure 2. Extraction method

IV. Extraction and Insertion of Choreography Fragments

Based on the formal definition of choreography fragments, in the following we present a method for their extraction and insertion from and to choreography models, respectively. Note that our approach is independent of the granularity of the choreography model, i.e., the choreography models to be extracted from or inserted into may also contain business related activities and variables besides the communication related activities and variables. For both the extraction and insertion the user decides which elements, public or private, are considered.

A. Extraction

Fig. 2 shows the steps of our method for extracting choreography fragments. First, a human user selects the elements of a choreography model to be extracted for later reuse (*Element Selection* step). The actual extraction of the choreography fragment graph is conducted in the *Extraction* step. Here, the selected elements are copied from the original model graph to a newly created choreography fragment. We aim to preserve a valid choreography model for each extracted fragment in order to allow an easy graphical manipulation of the fragment with existing tools. This means that, if logic is extracted from a choreography participant, it is also stored in a newly created choreography participant to preserve a valid structure. Additionally, if previously (transitively) connected activities in the model graph inside a choreography participant get disconnected, we reconnect them to preserve the original semantics of the choreography model. The results of the Extraction step can be reviewed and changed in the *Repair* step. Here, a human modeler can decide if the reconnection of control flow was according to the modelers intention. The choreography fragment can be adapted and enriched with additional modeling elements in the *Adaptation* step. Finally, the fragment is made persistent in the *Storing* step.

Algorithm 1 shows the procedure to be used in the Extraction step. The main idea of the algorithm is the iteration over all elements in the set S which have been manually selected in the Selection step. Subject to the type of the selected element, an appropriate sub-procedure is invoked to handle the element. Note that not all sub-procedures are shown due to space limitations. After all selected elements have been handled appropriately, each participant located in the choreography fragment is traversed to identify and reconnect disconnected components in the process model graph (Algorithm 4).

Algorithm 2 summarizes the `handleActivity` sub-procedure used by Algorithm 1 for handling activities. Other selected elements such as variables or message links are handled in a similar way. Note that also the variables the activity reads from and writes to are considered in Algorithm 2. We use a origin function for finding the original model element of a copy in the choreography fragment. Since an activity can only be located inside a participant, the parent participant of a selected activity also has to be copied into the choreography fragment if not already there. This is achieved by Algorithm 3 (`copyParents`). It requires a choreography model $\mathfrak{C}$, a choreography fragment $\mathfrak{F}_c$, an element *curr*, and a selected element *initial* as input. As long as the current element has a parent element, i.e., a participant or participant set, *copyParents* is invoked recursively. When the highest level of the nesting hierarchy is reached, i.e., the current element does not have a parent element, the recursion stops and the fragment $\mathfrak{F}_c$ is assigned to the parent element. Subsequently, it is checked if the current element already has

Algorithm 1: extractChoreographyFragment

```
1  input  : Choreography Model 𝔈
2  output : Choreography Fragment 𝔉_c
3  begin
4      S, 𝔉_c ← ∅
5      𝔉_c ← create choreography fragment
6      π_1(𝔉_c) ← π_1(𝔈) + "Fragment"
7      S ← getSelectedElements (𝔈)
8      foreach s ∈ S do
9          if s is an activity then
10             handleActivity (𝔈, 𝔉_c, s)
11         end
12         else if s is a control connector then
13             handleControlConnector (𝔈, 𝔉_c, s)
14         end
15         else if  s is a ... then
16             ...
17         end
18         foreach p_f ∈ π_2(𝔉_c) ∪ ⋃ R_con(π_3(𝔉_c)) do
19             reconnectLinks (origin_P(p_f, 𝔈), p_f)
20         end
21         return 𝔉_c
22     end
23 end
```

Algorithm 2: handleActivity

```
1  input  : Choreography Model 𝔈, Choreography Fragment 𝔉_c,
           Activity a
2  begin
3      Activity a_f ← copyParents(𝔈, 𝔉_c, a, a)
4      Participant p_f ← parentOf(𝔉_c, a_f)
5      if ∄ i(a)_c ∈ π_2(p_f.F) | origin_V(i(a)_c, 𝔈) = i(a) then
6          π_2(p_f.F) ← π_2(p_f.F) ∪ deepCopy(i(a))
7      end
8      if ∄ o(a)_c ∈ π_2(p_f.F) | origin_V(o(a)_c, 𝔈) = o(a) then
9          π_2(p_f.F) ← π_2(p_f.F) ∪ deepCopy(i(a))
10     end
11 end
```

Algorithm 3: copyParents

```
1  input  : Choreography Model 𝔈, Choreography Fragment 𝔉_c,
            Element initial, Element curr
2  output : Element element
3  begin
4      Element parent ← parentOf(𝔈, curr)
5      if parent ≠ ∅ then
6          parent ← copyParents(𝔈, 𝔉_c, initial, parent)
7      end
8      else
9          parent ← 𝔉_c
10     end
11     if ∃ elem_c | isPartOf(elem_c, parent) = true ∧
       origin_e(elem_c, 𝔈) = curr then
12         return elem_c
13     end
14     else
15         Element curr_c = ∅
16         if initial = curr then
17             curr_c ← deepCopy(curr)
18         end
19         else
20             curr_c ← shallowCopy(curr)
21         end
22         parent + curr_c
23         return curr_c
24     end
25 end
```

a copy in the parent element by using the `isPartOf` sub-procedure. If so, the copy is returned, otherwise a copy has to be created, added to the parent element, and returned. The + operator is used to express the addition of a generic element to its appropriate set inside its parent element. In case the current element is identical with the initial element, a deep (full) copy is made because the element has been selected completely, otherwise a shallow copy is made containing no nested elements.

With the help of Algorithm 4 (`reconnectLinks`) each participant included in the choreography fragment is traversed to find graph components that have become disconnected due to the initial selection. The main idea of the algorithm is to reconnect disconnected components with existing or newly created control connectors and present the changes to the human modeler for approval or manual repair, respectively. The algorithm requires a choreography model $\mathfrak{C}$, an original participant p_{orig}[1], and a copied participant p_f as input. The algorithms starts from the set of *start activities* (cf. Definition 1) of p_{orig} and traverses its process model graph G. The traversal is conducted in a DFS manner with the help of a stack data structure. The stack stores pairs consisting of an activity a and a predecessor activity a_{pre} which has been selected in the graph G. When a pair is popped from the stack, it is evaluated if the current activity a has been marked as selected. If not, and additionally it has not been marked

[1]Which is determined using the origin function $origin_P : P_f \times \mathfrak{C} \to P$

Algorithm 4: reconnectLinks

```
1  input  : Choreography Model 𝔈, Participant p_orig, Part. p_f
2  output : Participant p_f
3  begin
4      foreach a_start ∈ π_5(p_orig.G) | a_start ∈ A_start do
5          Stack.push((a_start, ⊥))
6          while Stack ≠ ∅ do
7              Pair x ← Stack.pop()
8              a ← π_1(x)
9              a_pre ← π_2(x)
10             if a is selected then
11                 if a_pre ≠⊥ ∧∄l ∈ π_6(p_f.F) | l = (a_pre, a, t)
                   then
12                     L_1 ← ∀ l ∈ π_6(p_f) | l = (a_pre, ⊥, t) ∧
                       isReachable (a_pre, a, origin_L(l, 𝔈))
13                     L_2 ← ∀ l ∈ π_6(p_f.F) | l = (⊥, a, t) ∧
                       isBReachable (a_pre, a, origin_L(l, 𝔈))
14                     if L_1 ≠ ∅ ∧ L_2 = ∅ then
15                         π_2(l_1 ∈ L_1) ← a
16                         mark all l ∈ L_1 as affected
17                     end
18                     else if L_1 = ∅ ∧ L_2 ≠ ∅ then
19                         π_1(l_1 ∈ L_2) ← a_pre
20                         mark all l ∈ L_2 as affected
21                     end
22                     else if L_1 ≠ ∅ ∧ L_2 ≠ ∅ then
23                         π_2(l_1 ∈ L_1) ← a
24                         mark all l ∈ L_1 ∪ L_2 as affected
25                     end
26                     else
27                         create l = (a_pre, a, t)
28                         π_6(p_f.F) ← π_6(p_f.F) ∪ l
29                         mark l as affected
30                     end
31                 end
32                 a_pre ← a
33             end
34             if a is not marked as visited then
35                 mark a as visited
36                 E ← outgoingLinks (p_orig, a)
37                 foreach l ∈ E do
38                     Stack.push((π_2(l), a_pre)
39                 end
40             end
41         end
42     end
43     return p_f
44 end
```

as visited, new pairs containing the target of the current activity's outgoing links and the current selected predecessor activity are pushed on the stack. If, however, the current activity is marked as selected, it is checked if there exists a control flow link in the choreography fragment starting from the selected predecessor activity to the current activity. If not, the set of outgoing dangling control connectors (L_1) of the current predecessor and the set of incoming dangling control connectors of the current activity (L_2) are calculated that can be used to repair the disconnection. The calculation is conducted by evaluating if there exists a path between the

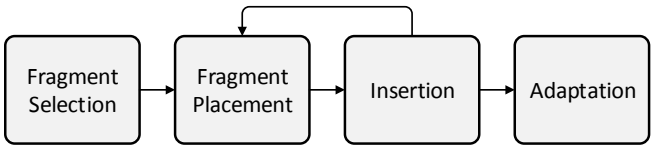


Figure 3. Insertion method

predecessor and the current activity in the original model via the dangling control connector at hand. For this, the sub-procedure `isReachable` is used to evaluate the path between the predecessor and the current selected activity, whereas the sub-procedure `isBReachable` evaluates the path starting from the current selected activity to the predecessor activity following the dangling control connectors in a backwards manner. If both sets are empty, a new control connector is created, otherwise the first control connector found in the corresponding non-empty set is used to reconnect the predecessor activity a_{pre} to the current selected activity a. Subsequently, activity a becomes the new predecessor activity a_{pre}. When a dangling link is used to reconnect activities, its transition condition is not considered. Thus, all links that would have been candidates for the reconnection are marked as *affected*, and the user has to check them after the extraction in the *Repair* step. If the transition condition is not appropriate, the user has to manually remove or change it. We denote this with *manual repair*. The algorithm terminates when all activities in the graph have been visited.

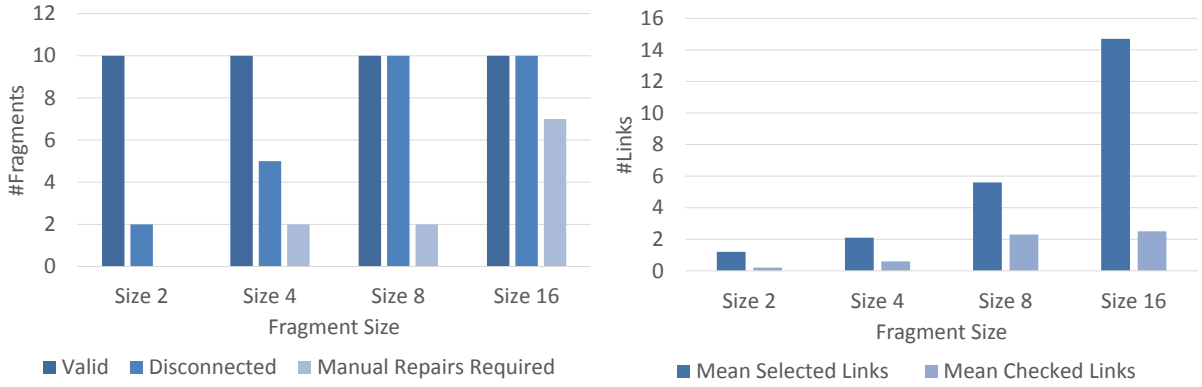
B. Insertion

Our proposal for semi-automatically inserting a choreography fragment into a choreography model is summarized by Fig. 3. With the help of a graphical modeling tool, a modeler first selects a fragment from a palette containing a list of previously extracted fragments (*Fragment Selection* step). Subsequently, the user places the desired fragment elements onto the choreography model (*Fragment Placement* step) by means of a choreography editor. This triggers the *Insertion* step. If choreography fragment elements are placed on existing elements, the user should be guided through the insertion of activities, control connectors, and variables by e.g., a graphical wizard indicating conflicts between already existing model elements. The user decides about keeping or replacing (name) conflicting elements. Furthermore, the wizard can help to connect dangling control connectors to other model elements or delete them. The surrounding participant structure is discarded when inserting into an existing participant in the choreography model. If an existing participant already contains activities, the user indicates the place for insertion. In case a selected participant of a choreography fragment has attached message links, the message links become visible when the involved counterpart participant is also inserted into the choreography model. Dangling message links of a participant are immediately visible and can be connected to other participants in the choreography model. The Fragment Placement and Insertion step are repeated until the choreography fragment is integrated into the choreography model, however, the modeler may decide that not all parts of the choreography fragment have to be inserted. The last step is the *Adaptation* step. Here the user can alter the inserted fragment and continue with modeling the choreography. Insertion is therefore heavily dependent on an appropriate choreography modeling tool, as discussed in the following.

V. Evaluation

For purposes of evaluating our proposal, and in place of an exhaustive field evaluation, we first simulate the use of our choreography fragment extraction method by an unskilled choreography modeler. Subsequently, the extraction and repair algorithms performance is measured and analyzed. The simulation of the fragment extraction method is conducted on an Intel(R) Core(TM) i7-3520M @ 2.90GHz system equipped with 16 GB RAM. The "Incident Management" choreography model published in [11] is used as the basis of this procedure. The choreography model consists of 5 participants and 23 activities. Fragments of size 2, 4, 8, and 16 (measured in number of included activities) are selected randomly from the activities of the participants of the choreography model. For each size group, 10 choreography fragments are generated. Additionally, control flow connectors are included into the extracted fragments in the following manner. If a selected activity is directly connected to one of the other (randomly) selected activities in the fragment, the connecting control flow link is also selected for inclusion in the fragment. Otherwise, the outgoing control flow connectors of each activity are included into the choreography fragment with a probability of 0.5. The randomized inclusion of activities and control flow connectors simulates a selection by an unskilled choreography modeler who is not aware of the semantic relations of the activities. Furthermore, the randomization procedure selects choreography fragments where graph components inside the choreography participants are disconnected and have to be reconnected by Algorithm 4.

Figure 4a shows the properties of the evaluated fragments grouped by fragment sizes. All 40 randomly extracted fragments were found to be structurally valid. This means that their nesting has been preserved and all disconnections of the process graphs inside of choreography participants have been repaired successfully. The number of initially disconnected process fragments increases with the fragment size. In the group of size 2, there were only 2 initially disconnected fragments, in the group of size 4, 5 initially disconnected fragments, and in the groups of size 8 and 16 all extracted fragments were initially disconnected. The group of fragment size 2 had no fragments that had to be manually repaired, in the group of fragment size 4 and size 8, 2 fragments required manual intervention, and in the group of fragment size 16,



(a) Properties of the extracted fragments

(b) Control flow links results

Figure 4. Evaluation results for a random extraction of fragments

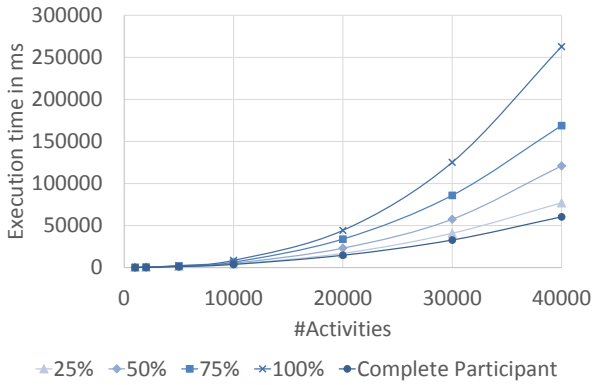


Figure 5. Extraction algorithm performance evaluation results

7 fragments had to be manually altered. The increase of fragments to be repaired can be attributed to the increasing number of activities in them, and especially to the number of control flow connectors in each choreography fragment. This becomes visible better when considering Fig. 4b showing the average number of randomly selected control flow connectors in each fragment size group and the average number of links that had to be manually checked. In the group of size 16, an average of 14.7 control flow connectors per fragment have been selected. In this group, it is more likely that dangling control flow connectors are part of the selection that are connected to other selected activities to repair a disconnection. Overall, roughly 25% of the randomly selected fragments had to be manually repaired; however, only one link per fragment on average was affected in each case.

Fig. 5 shows the performance evaluation of our extraction algorithm in terms of execution time. The measurements were conducted on the same system as described above. Since the size of the previously used choreography model is not big enough to meaningfully measure execution time, we generated 7 choreography models (essentially, connected graphs) consisting of one participant and a set of connected activities, ranging from 1000 to 40000 activities. We measured the performance of our extraction algorithm in two cases: *a)* the choreography participant is selected, and thus, all of its nested activities and control flow connectors are selected too, and *b)* a randomly chosen set of activities in the four quantiles are selected. In the latter case no control flow connectors are selected in order to enforce the reconnection of activities by the algorithm. As it can be seen from Fig. 5, the computational complexity of our algorithm is quadratic, however, the number of disconnections increases in the quantiles, and this in turn increases the execution time.

With respect to evaluating our insertion method, we rely on a proof-of-concept implementation as part of a choreography designer based on the Eclipse IDE. Fig. 6 shows the realization of our choreography modeling tool, the ChorDesigner [12], and its support of the insertion method. Extracted choreography fragments can be displayed by clicking on the fragment entry in the palette. This causes the ChorDesigner to open a new window (right-hand side). The choreography fragment, or parts of it, can be selected and placed into the left-hand side model. This provides users already familiar with our choreography designer with an intuitive and efficient way to use fragment insertion. A thorough user-based evaluation of the enhanced ChorDesigner functionality in conjunction with the incorporation of the extraction algorithm is future work.

VI. Related Work

The concept of process fragments as an element of reuse in developing process based applications has been covered in several works [2], [4], [5], [13], [14], [15]. For example, Markovic and Pereira [13] introduce an π-calculus based approach for process fragments. Process fragments can be semantically annotated for querying or autocompletion of process models. In [16], the concept of process fragments is used to describe change patterns and change reuse for process-aware information systems. The authors provide the adaptation pattern “Extract Process Fragment to Sub Process”

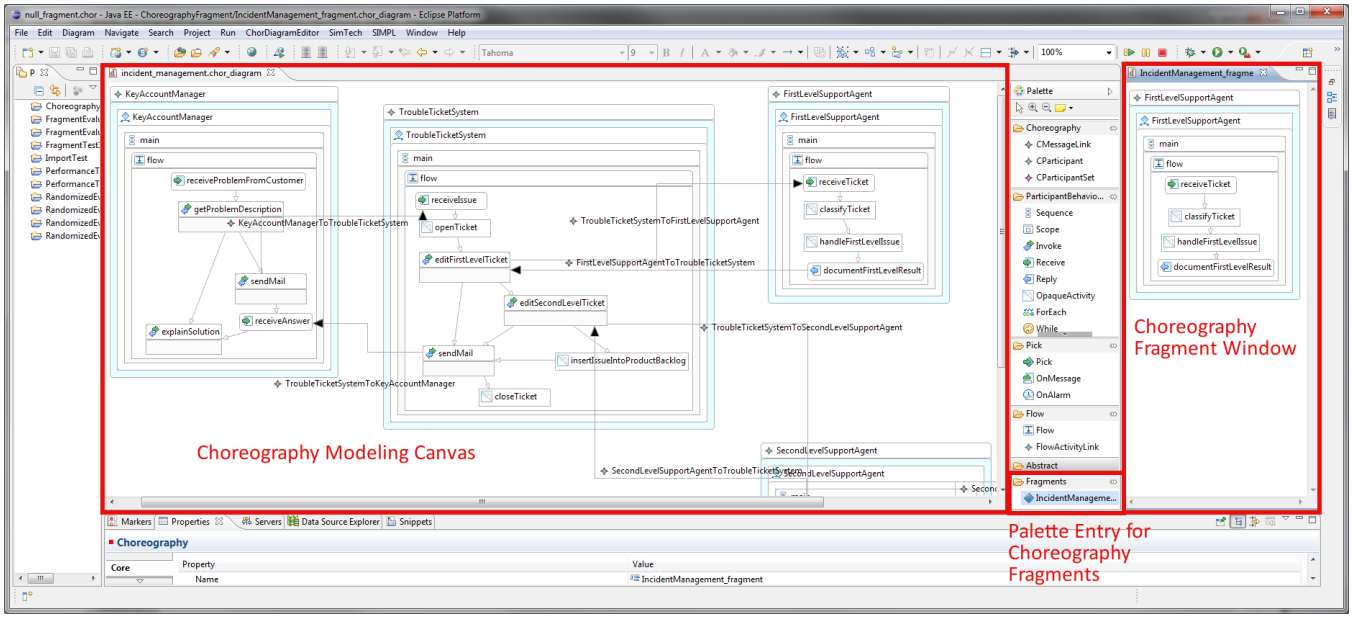


Figure 6. Fragment insertion example in the extended ChorDesigner

that describes the extraction and storing of process fragments in sub processes. However, no details on the actual algorithms for extraction are given. A similar approach is discussed in [2] in the context of refactoring techniques for large process model repositories. The "Extract Process Fragment" pattern can be used to reduce the size of a process model or remove redundant parts. Again, no specific details for the implementation of this approach are given. In [14], a signature tree based approach is used to query and reuse process fragments. Yang et al. [15] want to achieve the same goal by utilizing a context free grammar based approach.

The mentioned works lack a discussion of modeling elements that only exist on the level of choreographies such as participant sets or explicit message links connecting participants and how to extract them from existing models and insert them into new ones. This gap is closed with our notion of choreography fragments and the supporting methods and algorithms. Eberle et al. [5] introduce a formal model for process fragments and corresponding composition operations. We have adopted the formal model for process fragments and extended it to also consider choreography fragments. In [4], the authors propose the concept of a process fragment choreography by using process fragments to describe a collaboration scenario. The WS-CDL choreography language [17] has the Perform Activity construct that allows the recursive composition of existing choreographies into more complex choreographies. While the composition is also a method of reuse, our definition of choreography fragments not only comprises completely specified choreography models but also parts of a choreography model such as a message link connector to ease modeling for human users. Furthermore, our concept of choreography fragments is not tied to a specific choreography modeling language. Reuse of choreography models is also proposed by [18], who introduce federated choreographies, which are shared between different partners and may support other choreographies. The choreographies are realized by private orchestrations possibly contributing to more than one choreography. The main difference to our work is the specification of choreographies with interaction models. Montesi and Yoshida [19] propose so-called partial choreographies allowing the definition of choreographies where the implementation of some roles can be left open until run time. At run time, the partial choreographies are composed. The authors aim for the programming of choreographies using calculus, while we want to support choreography modeling with reusable elements extracted from choreography models.

VII. Conclusions And Future Work

In this work, we have introduced the concept of choreography fragments as elements of reuse in choreography modeling. Based on a formal definition of process models and fragments we have provided a formal model for choreography models and fragments. Choreography fragments represent parts of choreographies that can be reused across different choreography models in different domains. In order to support human choreography modelers with the (semi-)automatic extraction of choreography fragments from choreography models, the storing, and the insertion of choreography fragments into a new or existing choreography model we proposed a method and corresponding algorithms. Furthermore, we provided a method for the insertion of choreography fragments. Our

approach was evaluated for its efficacy and performance in an experimental manner (for fragment extraction), and by a proof-of-concept implementation (supporting the insertion method).

In future, we plan to further evaluate the usability of our approach and of the implemented prototype by means of user studies. Additionally, we will extend the choreography fragment concept in order to not only provide reuse during modeling time but also to enable flexibility during choreography enactment. Choreography fragments are one possibility to refine only partially specified choreographies during run time. For example, abstract constructs [8] could be replaced with choreography logic provided by choreography fragments. In order to support multi-tenancy scenarios [20], we plan to investigate how tenants and users can register choreography fragments to configure a template choreography model according to their requirements.

Acknowledgment

This work is funded by the projects FP7 EU-FET 600792 ALLOW Ensembles and the German DFG within the Cluster of Excellence (EXC310) Simulation Technology.

References

[1] F. Leymann and D. Roller, *Production Workflow - Concepts and Techniques*. PTR Prentice Hall, 2000.

[2] B. Weber and M. Reichert, "Refactoring Process Models in Large Process Repositories," in *Proceedings of CAiSE'08*. Springer, 2008, pp. 124–139.

[3] F. Gottschalk, W. M. P. van der Aalst, and M. H. Jansen-Vullers, "Configurable Process Models - A Foundational Approach," in *Reference Modeling*. Physica-Verlag HD, 2007, pp. 59–77.

[4] D. Schumm, D. Karastoyanova, O. Kopp, F. Leymann, M. Sonntag, and S. Strauch, "Process Fragment Libraries for Easier and Faster Development of Process-based Applications," *J. of Sys. Integration*, vol. 2, no. 1, pp. 39–55, 2011.

[5] H. Eberle, F. Leymann, D. Schleicher, D. Schumm, and T. Unger, "Process Fragment Composition Operations," in *Proceedings of APSCC'10*. IEEE, 2010, pp. 1–7.

[6] D. Schumm, F. Leymann, Z. Ma, T. Scheibler, and S. Strauch, "Integrating Compliance into Business Processes: Process Fragments as Reusable Compliance Controls," in *Proceedings of MKWI'10*. Universitätsverlag Göttingen, 2010, pp. 2125–2137.

[7] G. Decker, O. Kopp, and A. Barros, "An Introduction to Service Choreographies," *Information Technology*, vol. 50, no. 2, pp. 122–127, 2008.

[8] A. Weiß, S. Gómez Sáez, M. Hahn, and D. Karastoyanova, "Approach and Refinement Strategies for Flexible Choreography Enactment," in *Proceedings of OTM'14*. Springer, 2014, pp. 93–111.

[9] M. Sonntag and D. Karastoyanova, "Ad hoc Iteration and Re-execution of Activities in Workflows," *Int. J. On Advances in Software*, vol. 5, no. 1 & 2, pp. 91–109, 2012.

[10] G. Decker, O. Kopp, F. Leymann, and M. Weske, "BPEL4Chor: Extending BPEL for Modeling Choreographies," in *Proceedings of ICWS '07*. IEEE, 2007, pp. 296–303.

[11] Object Management Group, "BPMN 2.0 by Example Version 1.0 (non-normative)," 2011. [Online]. Available: http://www.omg.org/spec/BPMN/2.0/

[12] A. Weiß and D. Karastoyanova, "Enabling coupled multi-scale, multi-field experiments through choreographies of data-driven scientific simulations," *Computing*, pp. 1–29, 2014.

[13] I. Markovic and A. C. Pereira, "Towards a formal framework for reuse in business process modeling," in *BPM Workshops*. Springer, 2008, pp. 484–495.

[14] C. Zeng, Z. Lu, J. Wang, P. Hung, and J. Tian, "Variable Granularity Index on Massive Service Processes," in *Proceedings of ICWS'13*. IEEE, 2013, pp. 18–25.

[15] R. Yang, B. Li, J. Wang, L. He, and X. Cui, "SCKY: A Method for Reusing Service Process Fragments," in *Proceedings of ICWS'14*. IEEE, 2014, pp. 209–216.

[16] B. Weber, M. Reichert, and S. Rinderle-Ma, "Change patterns and change support features - enhancing flexibility in process-aware information systems," *Data Knowl. Eng.*, vol. 66, no. 3, pp. 438–466, 2008.

[17] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto. (2005) Web services choreography description language version 1.0. W3C. [Online]. Available: http://www.w3.org/TR/ws-cdl-10/

[18] J. Eder, M. Lehmann, and A. Tahamtan, "Choreographies as federations of choreographies and orchestrations," in *Advances in Conceptual Modeling-Theory and Practice*. Springer, 2006, pp. 183–192.

[19] F. Montesi and N. Yoshida, "Compositional choreographies," in *CONCUR 2013–Concurrency Theory*. Springer, 2013, pp. 425–439.

[20] M. Hahn, S. Gómez Sáez, V. Andrikopoulos, D. Karastoyanova, and F. Leymann, "SCE^{MT}: A Multi-tenant Service Composition Engine," in *Proceedings of SOCA'14*. IEEE, 2014, pp. 89–96.