Configurable Analysis of Sequential Multi-Party Choreographies

Andreas Schönberger and Guido Wirtz
Distributed and Mobile Systems Group
University of Bamberg
Bamberg, Germany
$\{$*andreas.schoenberger*$|$*guido.wirtz*$\}$*@uni-bamberg.de*

***Abstract*—For Business-To-Business integration (B2Bi) scenarios, the application of choreography and orchestration technology has become a core technique for resolving discrepancies between the interaction logic of individual partners and the intended overall message flow. While orchestrations govern the message exchanges of each single partner, choreographies define constraints and requirements for the message flow between all partners. Using choreographies, B2Bi scenarios can be analyzed from a global perspective before the business services of the integration partners for implementing orchestrations are developed.**
So far, B2Bi choreographies mostly have been binary, i.e., performed by exactly two partners. This paper shows how multi-party B2Bi choreographies can be composed from binary choreographies, how the multi-party perspective lends itself to attacking the so-called *partial termination* problem in a configurable way and how projections for the individual partners can be derived. Additionally, an algorithm for merging multiple projections of the same partner as well as rules for further reducing projections are given.

***Keywords*-multi-party choreographies; business services; business process management; B2Bi; ebXML BPSS**

I. Introduction

The choreography-orchestration tool-chain is a natural candidate to be applied to Business-to-Business integration (B2Bi). Choreographies may be used for modeling the interactions between integration partners from a global point of view and thus for defining and agreeing upon the business documents to be exchanged and the sequence of these exchanges. In a second step, each integration partner can leverage orchestration technology for implementing its obligations defined by the choreography. Thus, defining and analyzing choreographies has become an important method for defining the requirements and setting the context of the implementation of integration services. In the B2Bi domain, many approaches ([1], [2], [3], [4]) focus on strictly binary choreographies, i.e., on interactions between exactly two integration partners. While binary choreographies cover the majority of current B2Bi scenarios, multi-party scenarios actually are an implication of the concepts of supply chains/supply networks. Consequently, Huemer and Hofreiter argue [5] that interactions with more than one business partner at least have to be defined locally. Moreover, there are some real world examples that are not binary. For example, RosettaNet, a leading supply chain community of the ICT industry, defines the so-called "Order-To-Cash With Logistics Service Provider Scenario"[1] depicted in figure 1. In this scenario, a Customer, a Supplier and a Logistics Service Provider (LSP) role (represented by BPMN pools) are using RosettaNet PIPs (visualized as small cuboids labeled 3A4, 3A8 and so on) for exchanging business documents. Moreover, the local actions of each role for processing the business documents exchanged via PIPs are given. However, figure 1 only describes the intended flow of interactions and leaves out what happens if communication errors occur or if, for example, the Supplier and LSP role are not able to agree upon the provision of transportation services. Note that such technical/business errors only affect two of the three roles immediately (send and receive actions are defined for one role only). This raises the question whether or not erroneous behavior may have an effect on the remaining role and how to detect problematic execution paths.

In order to provide a widely applicable solution to this problem, this paper defines how multi-party choreographies can be composed from existing binary choreographies. Further, the negative effect of technical/business errors between two partners[2] on the remaining partners is captured as the so-called *partial termination* problem and a configurable algorithm for identifying problematic execution paths is sketched. A second algorithm for deriving role projections from multi-party choreographies is given for fostering straightforward systems development. Additionally, an algorithm for merging multiple projections (if existent) of the same role is given as well as rules for simplifying projections. This paper is an extended version of [6] and the main new material is the following:

- Configuration options are added to the algorithm for analyzing the partial termination problem in order to distinguish between in-

[1]http://www.rosettanet.org/Support/ImplementingRosettaNetStandards/eBusinessProcessScenarioLibrary/OrdertoCash/tabid/3320/Default.aspx

[2]The terms 'partner' and 'role' will be used interchangeably

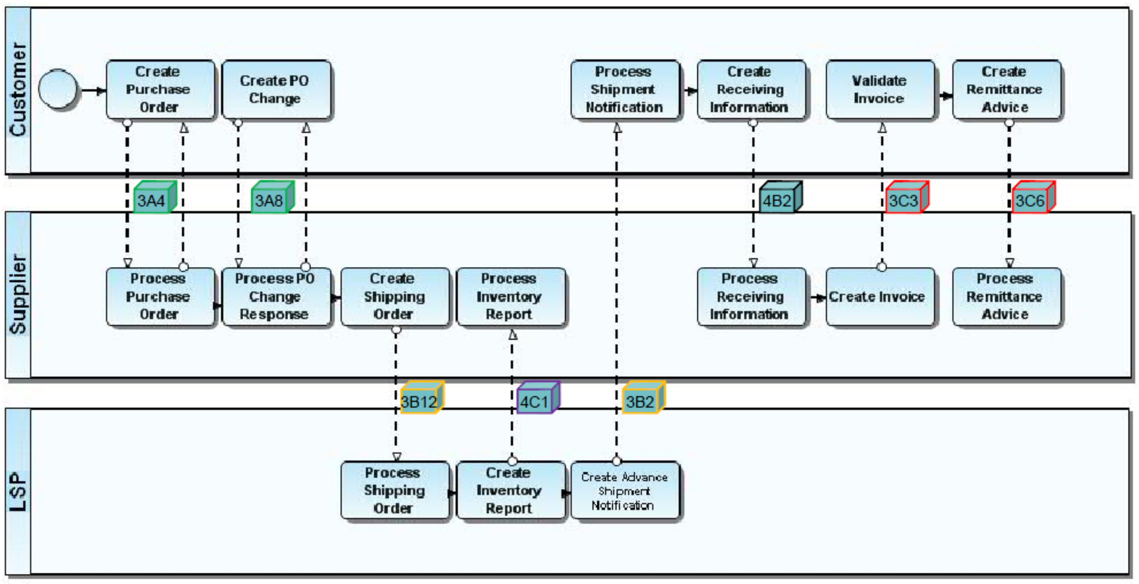

Figure 1. RosettaNet Order to Cash with Logistic Service Provider Scenario (from RosettaNet[1])

teractions after which roles still expect to participate again and those after which they do not.

- The algorithm for deriving role projections is modified such that the projection of a state is independent of the path through which the state is discovered.
- An additional algorithm for merging multiple projections of the same role is provided and the behavioral equivalence between the merge of projections and the set of individual projections is proven. This algorithm is complemented by rules for further reducing projections.

The paper proceeds as follows: Section II pins down the notion of B2Bi choreography used here. Section III introduces sequential multi-party (SeqMP) choreographies as new class of multi-party B2Bi choreographies. Also, the *partial termination* and *role projection* problems are introduced and their solutions are described. Finally, section IV discusses related work and section V concludes and points out directions for future work.

II. B2Bi CHOREOGRAPHIES

ebXML BPSS (ebBP, [7]) is the leading B2Bi choreography interchange format. The availability of domain specific concepts such as support for community-defined business document libraries or B2Bi Quality-of-Service (QoS) parameters like security and reliability make ebBP particularly useful for capturing the specification of B2Bi scenarios. ebBP concentrates on the interactions between integration partners, i.e., the sequence and types of business document exchanges (e.g., PIP cuboids in figure 1) and not on the local activities of integration partners for sending, receiving and processing these (e.g., activities within the BPMN pools in figure 1). In the terminology coined in [8], this corresponds to the specification of *interaction* choreographies instead of *interconnection* choreographies. For the purpose of the work at hand, it is sufficient to know that the concept of ebBP *BusinessCollaborations* (BC) can be used to capture the choreography of business document exchanges (the PIPs of figure 1) between two or more integration partner roles. ebBP *BusinessCollaborationActivities* (BCA) can be used to specify the execution of an existing BC within another BC, i.e., for composing BCs from existing BCs. This work builds upon the concept of binary BCAs, i.e., BCAs with exactly two roles. These are performed subsequently according to some guards that distinguish between the results of a BCA. As long as these concepts are available, this work is also applicable to other interaction-style choreography languages.

Note that ebBP is an XML-based B2Bi choreography interchange format and therefore needs a visual language to be useful for B2Bi modeling. The UN/CEFACT Modeling Methodology (UMM, [9]), the Business Choreography Language (BCL, [10]) as well as BPMN 2.0 choreographies ([11], section 11) are visual B2Bi choreography languages.

Figure 2 shows the use case of figure 1 in BPMN choreography notation. The use case is remodeled as a series of binary BCAs that are composed of 1 to 3 PIPs. The binary choreographies (BCAs) are modeled as so-called BPMN *Collapsed Call Choreographies* and visualized as rounded rectangles with a '+' at the bottom. The two bands at the top and at the bottom contain the integration partner roles participating in the call choreographies. The text in the middle contains an id (c1...c4), a name and the PIP types contained in the call

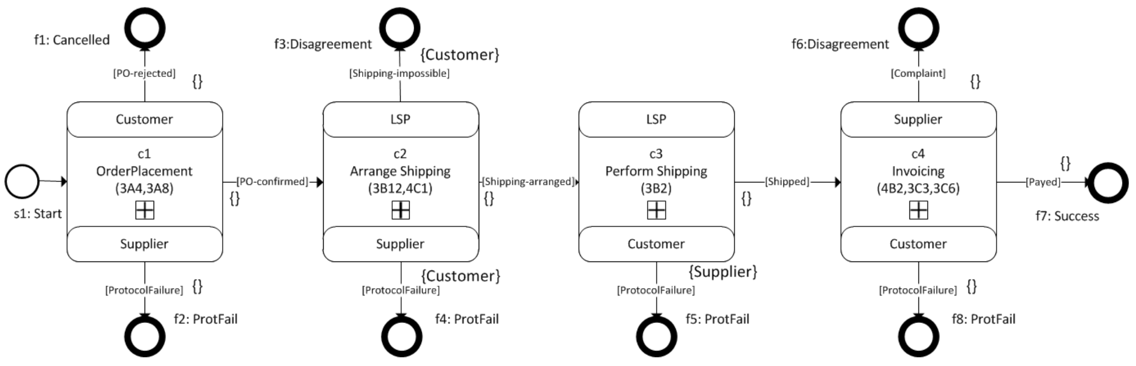

Figure 2. SeqMP model of the RosettaNet use case

choreographies (3A4, 3A8 and so on). Using the PIP types, it is easy to identify which call choreography corresponds to which part of the original RosettaNet choreography definition of figure 1. For defining the detailed structure of each binary call choreography, existing approaches are ready for use (cf. [4], [12], [13]).

III. Seq-MP Choreographies

This section first motivates the class of *sequential multi-party* (SeqMP) choreographies and gives its formal definition in subsection III-A. Subsection III-B then identifies two important problems in SeqMP choreographies and subsection III-C provides algorithms for solving these.

A. Definition

The class of SeqMP choreographies is tailored to the needs of B2Bi. By analyzing 100 scenarios of the publicly available *RosettaNet implementation guidelines* (for implementing B2Bi processes), we have discovered that the majority of interactions is binary (84 scenarios), i.e., are performed between exactly two integration partners. This is in line with academic research (cf. section I). The remaining multi-party interactions of our analysis can be split up into binary interactions. We have identified two factors that foster decomposability into binary interactions. First, the atomic building blocks of many B2Bi processes are binary transaction-like concepts for the exchange of request business documents and optional response business documents. In the case of RosettaNet, these atomic building blocks are called Partner Interface Processes (PIP) and despite the simple structure of PIPs the economic value exchanged using PIPs is worth billions of dollars (RosettaNet Standards Assessment 2008[3]). Similar 'atomic building blocks' can also be found in ebBP, UMM or BCL. Second, the control flow defined typically is fairly simple, i.e., does not apply concepts like parallel structures or hierarchical decomposition. This, in turn, is in line with a multi-case study of Reijers et al. [14] who report the results of an investigation of 16 business processes from six Dutch organizations: *"Business processes turned out to be completely sequential structures. Their routing complexity was only determined by choice constructs and iterations."* This finding is also backed by the B2Bi models created for the eBIZ-TCF project (http://www.moda-ml.net/moda-ml/repository/ebbp/v2008-1/en/, 10/24/2010) that do not use concurrent behavior.

Now, as control flow of B2Bi interactions tends to be simple and the atomic building blocks are binary, multi-party choreographies can be viewed as sequences of binary choreographies, i.e., as binary BCAs.

The advantage of using binary BCAs as building blocks for multi-party choreographies is that integration partners can be assumed to have agreed upon the result of the binary BCA. Also, both partners start and terminate the BCA more or less in lock-step. Consequently, the result of the binary BCAs can be used for routing the control flow of the multi-party choreography. In figure 2, this is indicated by guards (expressions placed in brackets) that are attached to the transitions. The guard `[PO-confirmed]` after BCA `c1` captures confirmation of the purchase order exchanged whereas `[PO-rejected]` captures rejection. The corresponding transitions of these guards link to BCA `c2` or end state `f1` accordingly. The annotations in curly braces are explained later. This concept of defining multi-party choreographies as sequentially performed binary BCAs with branching structures for defining control flow is reflected in the following definition.

Definition 3.1 (SeqMP Choreography):
A SeqMP choreography is a directed graph SeqMP (s_0, F,SBCA,T,R,RA) with the following elements:

- s_0 the (unique) start state.
- F a non-empty set of final states.
- SBCA a non-empty set of binary BCAs.
- T the union of the following transition sets
  - $T^{start} = \{(s_0,\text{true},\text{bca})\}$, bca $\in$ SBCA

[3]http://www.rosettanet.org.my/Download/2009 ImplementationStatistics 05.26.09.pdf

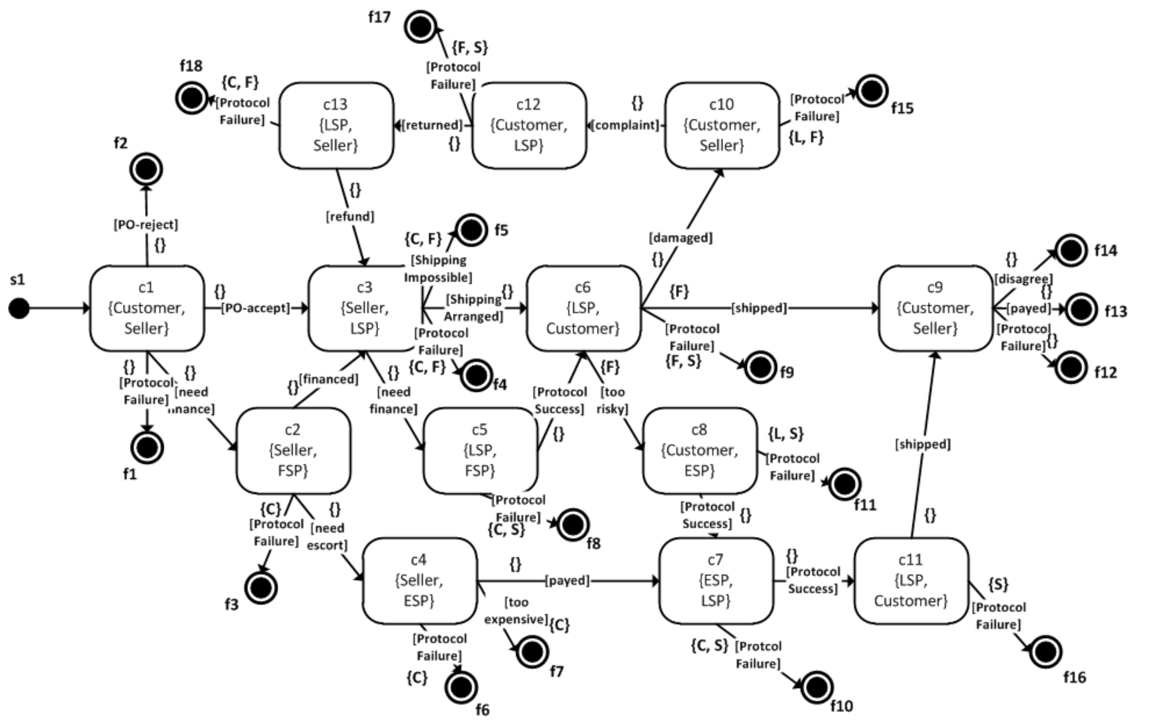

Figure 3. Seq-MP model of a complex use case (conflated visualization)

- $T^{end} \subseteq$ SBCA × G × F
- $T^{flow} \subseteq$ SBCA × G × SBCA

where G is a set of boolean guards consisting of the constants {true, else} and any disjunction of terms that consist of the names of the possible results of the BCA just performed. A term is evaluated upon termination of a BCA and becomes true when the BCA produces the corresponding result. 'else' becomes true if all guards of all other transitions with the same source become false.

- R the set of roles of the SeqMP process.
- RA: SBCA → R^2, a role assignment function that assigns exactly two roles to each BCA. □

Further, the following auxiliary functions are defined.

Definition 3.2 (SeqMP Auxiliary Functions):

- .-notation/#-notation is used for accessing the components of a tuple by name/index.
- *namesB* the function that computes the names of the results of a BCA.
- *namesG* the function that computes the names contained in a guard.
- A path *path*(a,b) between two nodes a,b ∈ $\{s_0\} \cup F \cup$ SBCA is a sequence of nodes a,$n_{1..x}$,b such that for all i=1...x-1, $(n_i,g_i,n_{i+1}) \wedge (a,g_a,n_1) \wedge (n_x,g_x,b) \in T$. The length of a path(a,b) *length*(path(a,b)) is the number of nodes in the sequence. Let Path(a,b) be the set of all paths between a and b. □

Based on this definition, it is possible to characterize the validity of SeqMP processes using the following three conditions.

Definition 3.3 (Valid SeqMP Choreography): A SeqMP choreography smp is valid iff the following three conditions hold:

1) **Subsequent role participation**:
$\forall (s_1, g, s_2) \in smp.T^{flow}$: $RA(s_1) \cap RA(s_2) \neq \emptyset$, i.e., for two subsequent BCAs at least one of the assigned roles must be the same (thence enabling synchronization between terminating one BCA and starting the next).
2) **Guard validity**:
Let $SUCC_{bca} \subseteq$ smp.T be the set of outgoing transitions from some bca ∈ smp.BCA with: ∀ t (t#1, t#2, t#3) ∈ $SUCC_{bca}$: t#1 = bca. Then, the guards of bca are valid iff:
($|SUCC_{bca}|$ = 1 ∧ for {t} = $SUCC_{bca}$: t#2 = true) ∨ (∀ t ∈ $SUCC_{bca}$: t#2 ≠ true ∧ (($\bigcup_{t \in SUCC_{bca}}$ namesG(t#2) = namesB(bca))∨ ($\bigcup_{t \in SUCC_{bca}}$ namesG(t#2) ⊂ namesB(bca) ∧ ∃ t1 ∈ $SUCC_{bca}$: t1#2 = else ∧ ∄ t2 ∈ $SUCC_{bca}$, t1 ≠ t2: t2#2 = else)) ∧ ∀ t3, t4 ∈ $SUCC_{bca}$, t3 ≠ t4: namesG(t3#2) ∩ namesG(t4#2) = ∅)
3) **Connectedness**:
(∀ f ∈ smp.F: Path(smp.s_0, f) ≠ ∅) ∧ ∀ bca ∈ smp.SBCA: Path(smp.s_0, bca) ≠ ∅ ∧ ∃ f ∈ smp.F: Path(bca, f) ≠ ∅. □

Actually, it would not be hard to extend this definition to using multi-party BCAs as building blocks (and even the algorithms in section III-C would work) as long as an agreed-upon result among all participants of the BCAs would be guaranteed. This, however, does not seem to hold true for many real-world scenarios.

B. Problems in Multi-Party Choreographies

Remodeling the RosettaNet Order-To-Cash use case as depicted in figure 2 immediately reveals two important problems, *partial termination* and creation of *role projections*.

Partial termination becomes obvious when looking at the transitions that lead into final states `f3`, `f4` and `f5` of figure 2. As the source states of these transitions are binary BCAs, only those roles participating in the respective BCA will be aware of the termination of the overall SeqMP choreography. However, the *Customer* role (in case of BCA `c2` *Arrange Shipping*) or the *Supplier* role (in case of BCA `c3` *Perform Shipping*) may still wait for some interaction to happen. This may not be a problem in case the individual BCAs of a SeqMP choreography are independent of each other, but the business semantics of RosettaNet's Order-To-Cash scenario contradicts independence of, for example, `c1` and `c4`. One possibility to attack this problem is adding additional BCAs implementing exception handling routines. However, modeling such multi-party choreographies may be hard because such exception handling BCAs may fail as well and suitable business documents for communicating exception handling semantics are not always available in business document libraries. Further, a business level problem like disagreeing on the conditions of shipping between *Supplier* and *LSP* may require business escalation routines between *Supplier* and *Customer* that are not intended to be implemented using business-document based choreographies. In essence, not defining explicit handling routines for arbitrary exceptional circumstances is a natural thing in process specification and hence partial termination is an integral problem of multi-party processes. Note that the different representations of the same use case in figures 1 and 2 do not have an influence on the actual existence of the partial termination problem. It is just not as obvious in the original RosettaNet representation of figure 1 because only the *intended* flow of interactions is modeled.

Creation of *role projections* is an obvious problem when considering that some roles may not participate in every BCA of a SeqMP choreography. Hence, the possible sequences of BCA executions with participation of a particular role r must be derived from the overall choreography and the BCAs without participation of r must be suitably abstracted.

Both problems are not hard to solve if the use case is as simple as in figure 2. However, more complex SeqMP choreographies such as the artificial use case depicted in figure 3 may be more challenging. The use case depicts a multi-party order-to-cash choreography between a *Customer*, a *Seller*, a *Logistics Service Provider* (LSP), a *Financial Service Provider* (FSP) and an *Escort Service Provider* (ESP). Due to space limitations, the BPMN choreography notation has been conflated in an ad-hoc manner by only showing the participating roles and an id for each call choreography. The business semantics of the use case may be derived to some extent from the guards on the transitions, but, for the purpose of this paper, the control flow of the use case is decisive. One striking observation is that *partial termination* is not a problem associated to final states only, but actually to transitions that partition a SeqMP choreography in parts with or without possible participation of a particular role. For example, by firing the transition between *c6* and *c8* as depicted in figure 3, there is no possibility that the *FSP* role will become active in the particular SeqMP instance anymore whereas participation still would be possible in *c6*.

C. Seq-MP Algorithms

This section presents algorithms for dealing with the *partial termination* and the *role projection* problem identified in the last section. Note that the algorithms are defined for SeqMP choreographies which are defined for binary BCAs as building blocks. However, the algorithms themselves also would work for multi-party BCAs as building blocks. For validation, the algorithms for calculating escalation sets as well as role projections have been implemented for an abstract model of SeqMP in Java. The prototype implementation together with the test graphs for figure 2 and 3 are available[4].

1) Computing Escalation Sets: As really safe multi-party choreographies that specify exception handling logic in full (cf. above) are hard to design, we propose the computation of so-called *escalation sets* for tackling the *partial termination* problem. Intuitively, escalation sets are sets of participation expectations that cannot be fulfilled any more upon firing a particular transition. Consider the transition (c2-f4) of figure 2 again. The *Customer* has participated in BCA c1 and then waits for BCA c4 to begin. So, a participation expectation has been created for the Customer. When firing (c2-f4), this expectation cannot be satisfied any more, but it could in c2. Additionally, the Customer does not participate in c2 and so she is not informed about that. Hence, firing (c2-f4) creates a partial termination problem for the Customer and the string 'Customer' is added to the escalation set of (c2-f4) (represented as curly braces added to the transition in figure 2) to specify that fact. For analogous reasons, 'Customer' is added to (c2-f3) and 'Seller' is added to (c3-f5).

[4]http://www.uni-bamberg.de/pi/confSeqMP-Algorithms

To capture this intuition more precisely, we define the concepts of 'Expectation' and 'Escalation Assignment'.

Definition 3.4 (Expectation):
An expectation is a 2-tuple E(r,en) where r $\in$ smp.R of some SeqMP choreography smp and en is a name. □

Definition 3.5 (Escalation Assignment):
An escalation assignment is a function ESA: smp.T $\rightarrow 2^E$ that, for a SeqMP choreography smp and a set of expectations E, assigns to each transition t $\in$ smp.T a set of expectations se $\subseteq$ E. □

Then, informally, escalation sets can be characterized as follows:

If an expectation (r,en) may have been created on some path to BCA bca and if the expectation still may be satisfied by some reachable BCAs and transitions and if the expectation may not be satisfied any more by taking a particular outgoing transition of bca and if r ∉ RA(bca) (because r would have full information otherwise) then the expectation goes into the escalation set of that transition.

Note that this informal definition does not refer to the concrete path that is taken at run-time for determining expectations. Instead, all paths that may have been taken for reaching bca are considered. That means that the participants of a SeqMP choreography can find out at design time which transitions potentially suffer from the *partial termination* problem at runtime. Based on this information, they may agree on appropriate actions for dealing with this issue, e.g., informing their integration partners via mail or phone. In so far, escalation sets are a means for analyzing the partial termination problem in multi-party choreographies, but not a fully automatic solution to the problem. Further, when firing a problematic transition at runtime the defined escalation set has to be compared to the actual expectations created during that particular process instance (which could easily be implemented by some logging feature).

So far, the details of when expectations are created and when these are satisfiable have not been discussed. We have identified three distinct strategies (or modes), namely 'ALWAYS', 'SELECTED', and 'RESOLVABLE', for accomplishing this task. The escalation sets then can be computed according to the 'Escalation Set Computation' algorithm as specified in algorithm objects 1 and 2. Below, we discuss the different strategies:

Strategy ALWAYS:
This strategy basically assumes that an expectation is created whenever a role r participates in a BCA and that it is satisfiable as long as r may still participate in some future BCA. The escalation criterion is (relative to some SeqMP smp) for some t $\in$ smp.T:

ESA(t(t#1,t#2,t#3)) = {(r,'ALWAYS') | ($\exists$ path(s_1, t#1), $s_1 \in$ smp.SBCA, length(path) > 1: r $\in$ RA(s_1)) $\wedge$ ($\exists$ path(t#1, s_2), $s_2 \in$ smp.SBCA, length(path) > 1: r $\in$ RA(s_2)) $\wedge$ ($\forall$ path(t#3, s_3), $s_3 \in$ smp.SBCA, length(path) > 1: r $\notin$ RA(s_3)) }
The advantage of this strategy is that users can apply this strategy to a valid SeqMP choreography

Algorithm 1: Compute Escalation Set: Part1

```
input :
smp, a valid SeqMP choreography;
mode, the operating mode;
rsa, a role selection assignm. //SELECTED mode;
epa, an expectation assignm. //RESOLVABLE mode;
rea, a resolution assignm. //RESOLVABLE mode;
output :
esa, an escalation assignm.;
variables :
// Maps for capturing reachable states/transitions;
Map<State,Set<State>> mapStFwd, mapStBwd;
Map<State,Set<Trans>> mapTrFwd, mapTrBwd;
// Result map for mapping transitions to expectations;
Map<Trans,Set<Expectation>> mapEsc;

procedure: compExpect(State s) :
1  if mode = ALWAYS then
2      Set<State> bwds = mapStBwd.get(s);
3      return $\bigcup_{st \in bwds}$ RA(st) × {'ALWAYS'};
4  else if mode = SELECTED then
5      Set<Trans> bwdtr = mapTrBwd.get(s);
6      return $\bigcup_{t \in bwdtr}$ rsa(t) × {'SELECTED'} × t.id();
7  else if mode = RESOLVABLE then
8      Set<Trans> bwdtr = mapTrBwd.get(s);
9      return $\bigcup_{t \in bwdtr}$ epa(t);
10 end
11 end procedure

procedure: compResolve(Set<Expectation> resolveSet, State s, Trans t) :
12 Set<Expectation> theRes = new Set();
13 State search = s;
14 if t ≠ null then search = t #3;
15 foreach Expectation e in resolveSet do
16     if mode ∈ {ALWAYS,SELECTED} OR (mode = RESOLVABLE AND rea(e) = ∅) then
           // check resolution via role participation
17         Set<State> fwds = mapStFwd.get(search);
18         if t ≠ null then fwds.add(t #3);
19         Set<Role> roles = $\bigcup_{st \in fwds}$ RA(st);
20         if e #1 ∈ roles then theRes.add(e);
21     else if mode = RESOLVABLE AND rea(e) ≠ ∅ then
           // check resolution via reachable transitions
22         Set<Trans> fwdtr = mapTrFwd.get(search);
23         if t ≠ null then fwdtr.add(t);
24         if ∃ res ∈ rea(e) . res ⊆ fwdtr then
25             theRes.add(e)
26         end
27
28 end
29 return theRes;
30 end procedure // continue...
```

Algorithm 2: Compute Escalation Set: Part2

```
algorithm:
// continue...
foreach State s in in a traversal of smp do
    // Comp. states/transitions reachable in
       forward/backward direction; s is not part of the
       reachability set
    mapStFwd.put(s,compStateFwd(s));
    mapStBwd.put(s,compStateBwd(s));
    mapTrFwd.put(s,compTransFwd(s));
    mapTrBwd.put(s,compTransBwd(s));
end
// Actual escalation assignment calculation
foreach State s in in a traversal of smp do
    Set<Expectation> eSet = compExpect(s);
    // Remove entries of roles participating in s
    foreach (r,e) $\in$ eSet do
        if r $\in$ RA(s) then eSet = eSet $\setminus$ {(r,e)};
    end
    // Determine resolvable expectations from s onwards
    Set<Expectation> allRes =
    compResolve(eSet,s,null);
    // Compare allRes to resolvable expectations of
       outgoing transitions
    foreach Trans t $\in$ SUCCs do
        Set<Expectation> trRes =
        compResolve(allRes,null,t);
        if allRes = trRes then
            mapEsc.put(t,new Set());
        else
            Set<Expectation> noRes =
            copy(allRes);
            noRes = noRes $\setminus$ trRes;
            mapEsc.put(t,noRes);
        end
    end
end
return mapEsc;
```

as is, i.e., no additional configuration is needed. The disadvantage is that the strategy ignores that the execution of some BCAs does not create expectations whereas others do.
The result of applying this strategy is reflected in the use cases of figures 2 and 3 where the escalation sets are attached to each transition by enumerating the according roles in curly braces. For figure 3, only the initial letters of each role are given and the string 'ALWAYS' is left out for presentation purposes. For example, the escalation set {F} of the transition between `c6` and `c9` of figure 3 says for the Financial Service Provider (FSP) role that it may have participated in an instance of the depicted SeqMP graph (if `c2` or `c5` were on the path to `c6`), that the FSP role still could be triggered (via the path (`c6`,`c10`,`c12`,`c13`,`c3`,`c5`)) and that it cannot be triggered any more when the transition has been fired (because from `c9` no BCA with participation of FSP can be reached). This may or may not be an issue depending on whether or not the FSP considers each BCA `c5` instance as a completely independent business case. The LSP role, in turn, is not included in the escalation set between `c6` and `c9` because it knows from the `[shipped]` outcome and the global SeqMP model that there is no way of being involved in the current SeqMP instance any more.

For the next strategy the concept of 'Role Selection Assignment' is introduced:

Definition 3.6 (Role Selection Assignment): A role selection assignment is a function RSA: smp.T $\rightarrow 2^{smp.R}$ that, for a SeqMP choreography smp, assigns to each transition t $\in$ smp.T the set of roles sr $\subseteq$ {r| r $\in$ RA(t#1)} for which the expectation to participate once more during the choreography is created upon firing t. □

Strategy SELECTED:
This strategy is similar to the 'ALWAYS' strategy but not every outgoing transition of a BCA creates an expectation but only those transitions that the user configures by specifying a role selection assignment. Additionally, the user has the option to specify for which of the participating roles of a BCA a particular outgoing transition creates an expectation. Expectations are satisfiable as long as r may still participate in some future BCA. The escalation criterion is (relative to some SeqMP smp and role selection assignment RSA) for some t,t$'$ $\in$ smp.T:
ESA(t(t#1,t#2,t#3)) = {(r,'SELECTED'+t.id()) | ($\exists$ path(s$_1$, t#1), s$_1$ $\in$ smp.SBCA: r $\in$ RSA(t$'$) with t$'$#3 = s$_1$) $\wedge$ ($\exists$ path(t#1, s$_2$), s$_2$ $\in$ smp.SBCA, length(path) > 1: r $\in$ RA(s$_2$)) $\wedge$ ($\forall$ path(t#3, s$_3$), s$_3$ $\in$ smp.SBCA, length(path) > 1: r $\notin$ RA(s$_3$)) }
The advantage of this strategy is that it imposes a moderate configuration burden on the user while offering the possibility to exclude transitions from expectation creation. The disadvantage of this strategy is that the user has no control about the events that lead to the satisfaction of an expectation.

Figure 4 shows the result of applying the SELECTED strategy to the complex use case of figure 3 with the following role selection assignment RSA:
RSA = {(c1-c3,{customer, seller}), (c3-c6,{esp, seller}), (c4-c7,{lsp, seller})}
There are three major differences to observe when comparing this result to the result of the ALWAYS strategy as shown in figure 3: First, the Customer role is not included in the escalation sets of transitions (c2-f3), (c4-f6) and (c4-f7) because only transition (c1-c3) is configured to create an expectation for the Customer and this transition is not reachable in backward direction of those transitions. Second, as there are no expectations configured for the Financial Service Provider this role is not included in any escalation set. Third, multiple expectations are created for the Seller role. The transitions have been added

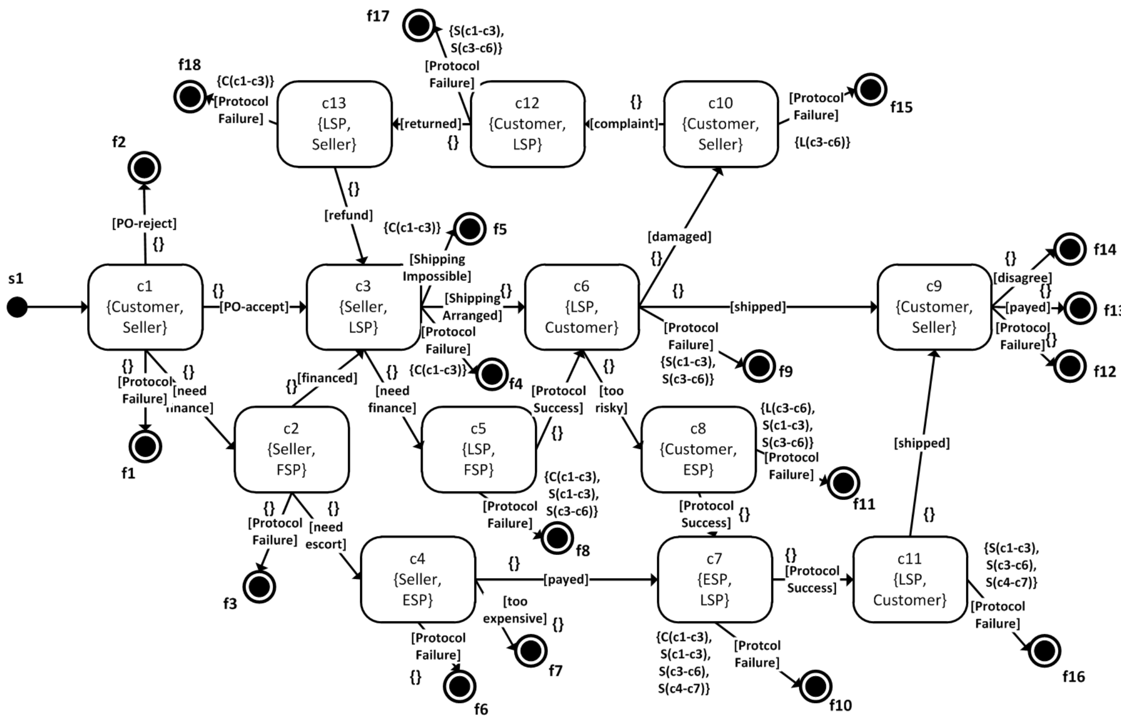

Figure 4. SELECTED Strategy Applied to the Use Case of Figure 3

to the role names in figure 4 to distinguish between the various expectations, e.g., ‘S(c1-c3)’, ‘S(c3-c6)’ or ‘S(c4-c7)’. The escalation sets then reflect which of those expectations potentially are violated. So, there are three violated Seller expectations for (c11-f16) whereas there are only two violated Seller expectations for (c12-f17). Note that these results highly depend on the role selection assignment provided by the user and therefore the business meaning of escalation sets is heavily influenced by the user. For example, not configuring an expectation for the FSP role may mean that the BCAs with FSP participation are completely independent of other BCAs or it may mean that the user just did not want to focus on the FSP role.

For the last strategy, the definitions of ‘Expectation Assignment’ and ‘Resolution Assignment’ are needed:

Definition 3.7 (Expectation Assignment): An expectation assignment is a function EPA: smp.T $\rightarrow 2^{E}$ that, for a SeqMP choreography smp and a set of expectation names EN, assigns to each transition t $\in$ smp.T the set of expectations se $\subseteq$ E=$\{$(r,en)$|$ r $\in$ RA(t#1), en $\in$ EN$\}$ that is created upon firing t. □

Definition 3.8 (Resolution Assignment): A resolution assignment is a function REA: E $\rightarrow 2^{2^{smp.T}}$ that, for a SeqMP choreography smp and a set of expectations E, assigns to each expectation e $\in$ E the set of sets of transitions setRes $\subseteq 2^{smp.T}$ for which each element resolves e. □

Strategy RESOLVABLE:
The last strategy offers the possibility to create multiple expectations per role upon firing a transition and to define sets of sets of transitions that need to be fired to satisfy an expectation. The escalation criterion is (relative to some SeqMP smp, expectation assignment EPA and resolution assignment REA) for some t,t′ $\in$ smp.T:
ESA(t(t#1,t#2,t#3)) = $\{$ep $\in$ EPA(t′) $|$ ($\exists$ path(s_1, t#1), $s_1 \in$ smp.SBCA: t′#3 = s_1) $\wedge$ ($\exists$ a sequence of transitions $t_1,...,t_n \in$ rp.T. (t_1#1 = t#1 $\wedge$ $\exists$ reSet $\in$ REA(ep). reSet $\subseteq t_1,...,t_n$)) $\wedge$ ($\nexists$ a sequence of transitions $t_1,...,t_n \in$ rp.T. (t_1#1 = t#3 $\wedge$ $\exists$ reSet $\in$ REA(ep). reSet $\subseteq t_1,...,t_n$)) $\}$
While this strategy places considerable burden upon the user to define comparatively complex expectation and resolution assignments, it allows for fine-grained configuration possibilities to define when expectations are created and satisfied. Additionally, users may choose to avoid the specification of complete expectation and resolution assignments by focusing in on only selected expectations and resolutions.

Figure 5 shows the result of applying the RESOLVABLE strategy to the complex use case of figure 3 with the following expectation assignment EPA and resolution assignment REA:
EPA =
$\{$(c1-c3,$\{$(Customer,ReceiveProduct), (Customer,PayProduct)$\}$),
(c1-c2,$\{$(Customer,EscortedProduct)$\}$),
(c6-c10,$\{$(Customer,ResolveDamage)$\}$),
(c10-c12,$\{$(Customer,SendBack)$\}$),

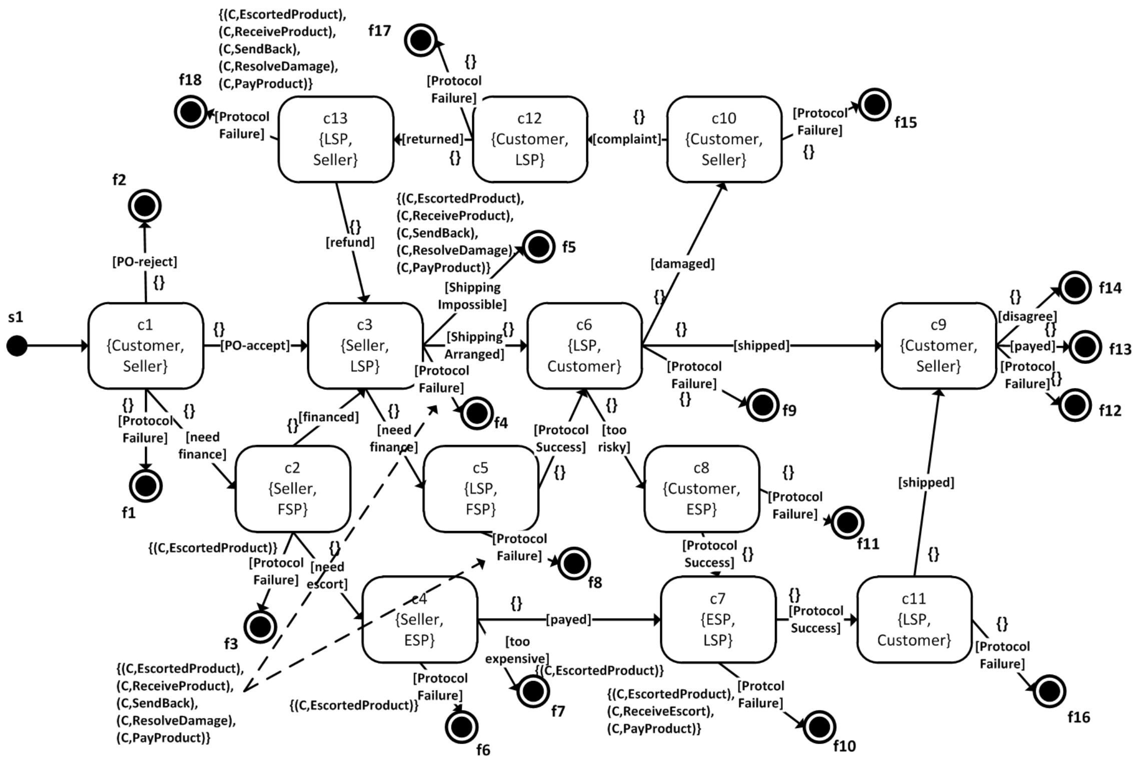

Figure 5. RESOLVABLE Strategy Applied to the Use Case of Figure 3

(c8-c7,{(Customer,ReceiveEscort)})}

REA = {((Customer,ReceiveProduct),{{(c6-c8)},{(c6-c9)}}), ((Customer,PayProduct),{{(c9-f13)}}), ((Customer,EscortedProduct),{{(c11-c9),(c9-f13)}}), ((Customer,ResolveDamage),{{(c6-c8)},{(c6-c9)}}), ((Customer,SendBack),{{(c12-c13)}}), ((Customer,ReceiveEscort),{{(c11-c9)}})}

When comparing the RESOLVABLE strategy to the first two strategies observe the following extensions: First, there may be multiple expectations per transition for the same role. For example, upon firing (c1-c3) the expectations named 'ReceiveProduct' and 'PayProduct' are created for the Customer role. Second, the resolution of expectations is configurable and does not depend on role participation during BCAs. For example, transition (c9-f13) is configured to resolve the PayProduct expectation whereas either (c6-c8) or (c6-c9) can resolve ReceiveProduct. Therefore, (Customer, PayProduct) is part of the escalation set of (c7-f10) whereas (Customer, ReceiveProduct) is not. The reason is that neither (c6-c9) or (c6-c8) are reachable in forward direction from BCA c7. So firing (c7-f10) does not cut off any resolution of ReceiveProduct and therefore ReceiveProduct does not go into the escalation set of (c7-f10). Note that the result of the RESOLVABLE strategy highly depends on the expectation and resolution assignment provided by the user and thus allows a fine-grained analysis of SeqMP choreographies.

2) Computing Role Projections: Role projections of multi-party choreographies are useful for focusing on the relevant behavior of a particular role *r*. Our approach for computing projections of a SeqMP *smp* for r is based on abstracting interactions without participation of *r* using so-called event-based choices (EBCs). So, if a BCA without participation follows a BCA with participation of r then the second BCA is replaced by an EBC. The EBC then is the new target of the transition between the two BCAs. Conversely, if a BCA with participation of r follows a BCA without participation then the transition between the two BCAs is removed and a new transition between the EBC for abstracting the former BCA and the BCA with participation of r is created. In figures 6, 7, 8 and 9, black vertical bars are used to represent EBCs. Consider figure 6 which shows the projection of the use case of figure 3 for the Customer role. As the Customer does no participate in BCA c3 it is abstracted by EBC ebc1 and transition (c1-c3) is linked to ebc1. c5 is then conflated with ebc1 because the Customer does not participate in c5 either. ebc1 then is linked to BCA c6 as this is the next BCA with participation of Customer. The guard of this transition is "true" because it is transparent for the Customer what happens in the meantime. Note that EBCs may also be created if the predecessor is not a BCA with participation of

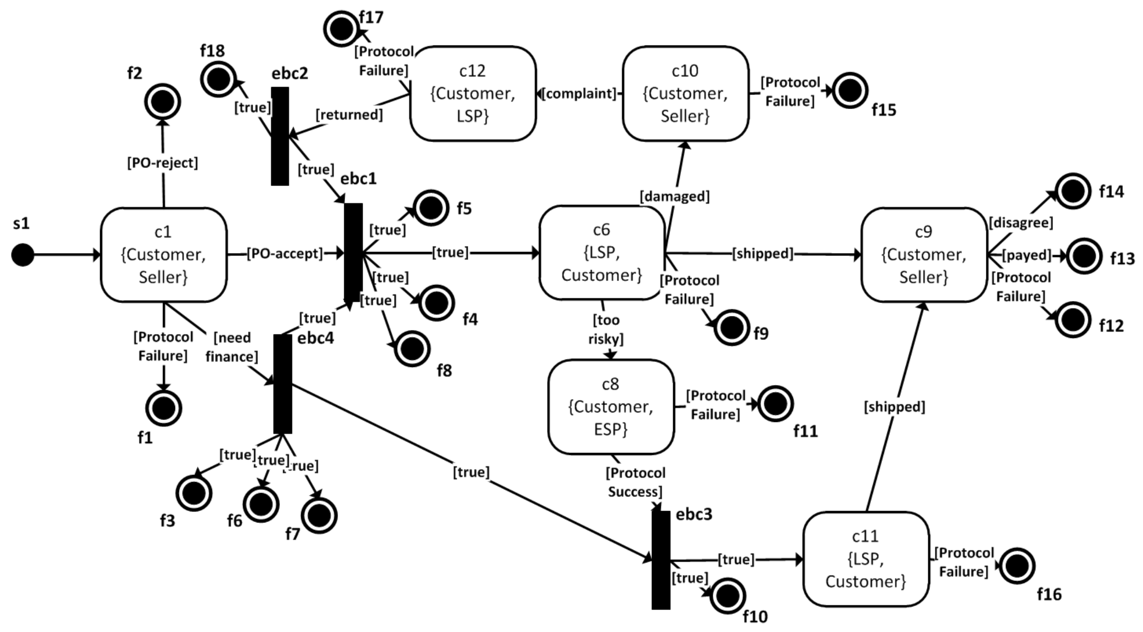

Figure 6. Projection for the Customer Role (Use Case of Figure 3)

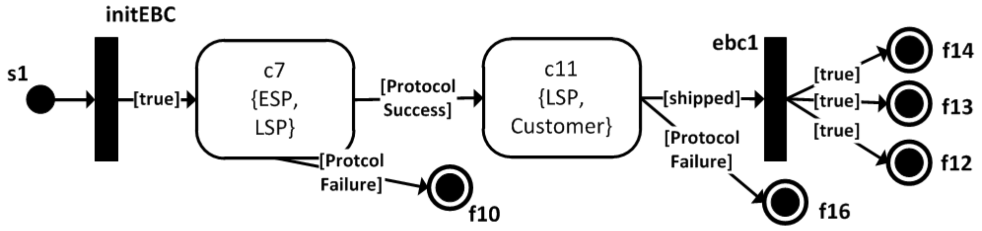

Figure 7. 1 out of 2 Possible Projections for the LSP Role (Use Case of Figure 3)

the focal role. For example, BCAs c2 and c4 are represented by ebc4. When processing transition (c4-c7) an extra EBC (ebc3) is created because c7 is reachable via c8 as well. Merging ebc4 and ebc3 would put the behavioral integrity of the projection at risk because then all outgoing transitions of ebc4 would be reachable via c8 as well. Note that in [6], we chose to merge EBCs in such a case and to simply introduce an additional EBC for the path via c8. We have changed the behavior of the algorithm because the new algorithm creates more compact projections and provides that the projection of a state is independent of the path through which it is discovered.

Definitions 3.9 and 3.10 formally capture role projections and some auxiliary functions.

Definition 3.9 (Role Projection):
A role projection rp_r for role r of a SeqMP choreography smp is a directed graph Proj (s_0, S,T) with the following elements:

- s_0 = smp.s_0 the (unique) start state.
- S = s_0 ∪ EBC ∪ F ∪ SBCA the states of the projection with EBC a set of event-based choice states, F ⊆ smp.F, SBCA ⊆ smp.SBCA with ∀ bca ∈ SBCA. r ∈ smp.RA(bca).
- T ⊆ S × smp.G × S the set of transitions where for each t ∈ T. t ∈ EBC × {true} × EBC ∨ t ∈ smp.T ∨ ($\exists t'$ ∈ smp.T. (t=(t'#1,t'#2,e) ∨ t=(e,true,t'#3)) ∧ e ∈ EBC). □

Definition 3.10 (Projection Auxiliary Functions): The auxiliary functions defined for SeqMP choreographies in definition 3.2 are correspondingly defined for role projections. In particular, a $path_r$(a,b) is a path between two nodes a and b in projection rp and $Path_r$(a,b) is the set of all paths between a and b in projection rp. If rp.s_0 links to an event-based choice then *rp.initEBC* can be used to refer to that event-based choice. □

The role projection algorithm presented in algorithm objects 3, 4 and 5 gives the details of computing role projections. Note that there may be more than one projection per role. We choose deliberately to allow for multiple projections of a particular SeqMP *smp* for a particular role *r* because different branches of a multi-party choreography potentially may comprise completely unrelated BCAs of r. Consider figure 3 and the LSP role. If an instance of the choreography began with BCAs c1, c2 and c4 then only the BCAs of figure 7 would be possible for the LSP role. Assume further there was no transition between c8 and c7. Then, depending on whether transition (c2-c4) or (c2-c3) is taken, completely disjoint sets of BCAs with LSP-participation are possible. In that situation, we believe that integration participants should not be forced to view unrelated BCAs as part of a single collaboration. We use algorithm 3 for identifying potentially disjoint projections for a particular role r. Once the first BCA with participation of r is found that could be the starting point of such a projection, the computation of a role-specific projection at a particular state is started (algorithm objects 4 and 5). For the case of the LSP role of the use case of figure 3, this approach leads to two individual projections that overlap. In this case, the individual projections still may be considered to be two different use cases or may be merged which depends on the system setting of the integration participant. Algorithm object 6 shows how two distinct projections of the same role can be merged such that the individual projections and the corresponding merge are behaviorally equivalent. The algorithm is defined for two projections, but it can be applied iteratively to cover the case of more than two projections.

Algorithm 3: Projection Computation

```
input     :
A valid SeqMP smp to be analyzed
output    :
A mapping of roles to their projections: smp.R → Set<State>
variables :
Set<State> computed;
//initially maps each role of smp to an empty set of projections;
Map<Role,Set<State>> projs;
Boolean processBegin;
algorithm:
processBegin = true;
foreach State s of each path without loop in a depth first traversal of smp do
    Set<Role> known = $\bigcup_{r\in}$states on the current path $\setminus$ s$^{\text{RA}(r)}$;
    Set<Role> curr = RA(s);
    curr = curr $\setminus$ known;
    if curr $\neq \emptyset \wedge$ s $\notin$ computed then
        foreach Role r in curr do
            // Add role projection for r starting at s
            projs.get(r).add(doProjection(r, s, processBegin));
        end
        computed.add(s);
    end
    if processBegin then processBegin = false;
end
return projs;
```

For reasoning about behavioral equivalence, definitions 3.11 and 3.12 introduce the concepts of 'Boundary Overlap Node' and 'Execution Traces'.

Algorithm 4: Role Projection - State s: Part 1

```
input     :
State s and Role r of a valid SeqMP smp;
boolean first, true if s is the first BCA in smp
output    :
The first state of the role projection $s_{out}$
variables :
Map<String, State> proj //state ids to projected states;
Map<State,State> visitMap //source states to projected states;
procedure: walk(State curr) :
State mSelf = visitMap.get(curr);
foreach Trans t =(t #1,t #2,t #3) in SUCC_curr do
    State mTarg = visitMap.get(t #3);
    if mTarg ≠ null then
        if mSelf is an EBCState AND mTarg ∉ SUCC_mSelf then //define transition between #1 and #3 with guard #2
            trans(mSelf, "true", mTarg);
        else
            trans(mSelf, t #2, mTarg);
        end
    else
        if t#3 is a FinalState ∨ r ∈ RA(t#3) then //copy state
            State nextCopy = t#3.scopy();
            visitMap.put(t#3, nextCopy);
            proj.put(t#3.id(), nextCopy);
            if mSelf is an EBCState then
                trans(mSelf, "true", nextCopy);
            else
                trans(mSelf, t#2, nextCopy);
            end
        else //abstract by event-based choice
            if mSelf is an EBCState then
                // alternative path to t #3?
                if ∃ bca ∈ smp.BCA. r ∈ RA(bca) ∧ Path(bca,t #3) ≠ ∅ then
                    EBCState ebc = new EBCState();
                    visitMap.put(t#3, ebc);
                    proj.put(ebc.id(), ebc);
                    trans(mSelf, "true", ebc);
                else
                    visitMap.put(t#3, mSelf);
                end
            else
                EBCState ebc = new EBCState();
                visitMap.put(t#3, ebc);
                proj.put(ebc.id(), ebc);
                trans(mSelf, t#2, ebc);
            end
        end
        walk(t#3);
    end
end
end procedure // continue...
```

Theorem 3.1 then formally captures behavioral equivalence and lemmata 3.1, 3.2 and 3.3 are used during its proof.

Definition 3.11 (Boundary Overlap Node): A boundary overlap node of two role projections rp_1 and rp_2 is a BCA bca such that bca $\in$ rp_1.SBCA $\wedge$ bca $\in$ rp_2.SBCA $\wedge$ (($\nexists bca'$ $\in$ rp_2.SBCA. $\exists$ path($rp_1.s_0$,bca'),

Algorithm 5: Role Projection - State s: Part 2

```
algorithm:
// ...continue
$s_{out}$ = new StartState("s1");
State currCopy = s.scopy() // copy s without transitions
proj.put("s1", $s_{out}$);
proj.put(s.id(), currCopy);
if first then
    trans (proj.get("s1"), "true", proj.get(s.id()));
else
    proj.put("ebc0", new EBCState("ebc0"));
    // define transition between #1 and #3 with guard #2
    trans ($s_{out}$, "true", proj.get("ebc0"));
    trans (proj.get("ebc0"), "true", proj.get(s.id()));
end
visitMap.put(s, currCopy);
walk(s);
return $s_{out}$;
```

Algorithm 6: Projection Merge

```
input   :
State $rp_1$, $rp_2$, two projections of the same role;
output  :
State $rp_m$, the merged role projection;

algorithm:
1  $rp_m$ = $rp_1$.deepCopy() // copy of complete projection of $rp_1$
   // insert extra leading event-based choice to avoid new traces by loops back to $rp_1$.initEBC
2  State mergeEBC = new() EBCState();
3  $rp_m$.S.add(mergeEBC);
   // use prj(state) to get the copy of state
4  $rp_m$.T.del((prj($rp_1$.$s_0$),"true",prj($rp_1$.initEBC)));

5  $rp_m$.T.add((prj($rp_1$.$s_0$),"true",mergeEBC));
6  $rp_m$.T.add((mergeEBC,"true",prj($rp_1$.initEBC)));

   // prepare adding $rp_2$
7  $rp_m$.T.add((mergeEBC,"true",$rp_2$.initEBC.copy()));

8  foreach Trans t =(t #1,t #2,t #3) in a breadth first traversal starting from $rp_2$.initEBC do
9      if t #3.id() $\notin$ $rp_m$.S.ids() then
10         $rp_m$.S.add(t #3.copy());
11     end
       // copy transitions
12     $rp_m$.T.add({(prj(t #1),t #2,prj(t #3))});
       // Stop upon encountering an overlap node
13     if t #3.id() $\in$ $rp_1$.S.ids() then
14         stopTraversalBranch();
15     end
16 end
17 return $rp_m$;
```

path(bca',bca). bca $\in$ path(rp$_1$.s$_0$,bca')) $\vee$ ($\nexists bca' \in$ rp$_1$.SBCA. $\exists$ path(rp$_2$.s$_0$,bca'), path(bca',bca). bca $\in$ path(rp$_2$.s$_0$,bca'))). $\square$

Definition 3.12 (Execution Traces):
An execution trace *trace*(a,b) between two states a and b of a role projection rp is a sequence of states a,st$_0$,...,st$_n$,b that is derivable from some path(a,b) by removing all event-based choices. Let Trace(a,b) be the set of all traces between a and b. For some state s $\in$ rp.S of projection rp, the set of *leading* traces is Trace^{lead}(s) = Trace(rp.s$_0$,s) and the set of *trailing* traces is Trace^{trail}(s) = $\bigcup_{\text{f}\in\text{rp.F}}$Trace(s,f). $\square$

Theorem 3.1 (Behavior Preservation of Merge): Consider two non-identical projections rp$_1$ and rp$_2$ for the same role r of a valid SeqMP choreography smp and the merge rp$_m$ of rp$_1$ and rp$_2$ as derived by algorithm 6. Then Trace^{trail}(rp$_m$.s$_0$) = Trace^{trail}(rp$_1$.s$_0$) $\cup$ Trace^{trail}(rp$_2$.s$_0$). $\square$

Proof 3.1:
The proof is split up into two parts:
Part 1: rp$_m$ produces all traces contained in rp$_1$ and rp$_2$.
By lemma 3.3 and the definition of execution traces, it is sufficient to consider the forward subgraphs of rp$_1$, rp$_2$ starting with rp$_1$.initEBC, rp$_2$.initEBC respectively. The subgraph starting with rp$_1$.initEBC is a subgraph of rp$_m$. So, rp$_m$ produces all traces of rp$_1$. Further, let B be the set of boundary nodes of rp$_1$ and rp$_2$. By algorithm 6, lines 9 to 12, all states and transitions not connected to a boundary node are copied to rp$_m$ without modification. Otherwise, consider an arbitrary path p = rp$_2$.initEBC,...,bca$'$,...,b with b $\in$ B. If bca$'$ $\in$ rp$_1$.SBCA then bca$'$ is not copied to rp$_m$ (line 9 of algorithm 6). However, by lemma 3.2, the forward graph of bca$'$ is already contained in rp$_m$.
Conversely, if bca$'$ $\notin$ rp$_1$.SBCA then bca$'$ $\notin$ Path(b$'$,f) for any b$'$ $\in$ B and f $\in$ smp.F because every forward reachable state is visited in a projection of b$'$. Then $\nexists$ bca$''$. ($\exists$ path(rp$_2$.initEBC,bca$''$),path(bca$''$,bca$'$) $\wedge$ bca$''$ $\in$ Path(b$'$,f) for any b$'$ $\in$ B and f $\in$ smp.F). Then all states and transitions visited on p between rp$_2$.initEBC and bca$'$ are copied to rp$_m$. $\square$

Part 2: rp$_m$ produces no trace that is neither contained in rp$_1$ nor rp$_2$.
For part 2, assume the opposite. Then, there would have to be some BCA or transition in rp$_m$ that adds to a new trace and that is neither contained in rp$_1$ nor in rp$_2$. During copying rp$_1$ (line 1), no new states or transitions are created. As long as no boundary node is encountered, copying transitions and states of rp$_2$ just reproduces rp$_2$. For a boundary node b however Path(rp$_2$.initEBC,b) is retained without structural modification in rp$_m$ and Trace$^{trail}_{rp_m}$(b) = Trace$^{trail}_{rp_2}$(b) by means of lemma 3.2 $\square$

Lemma 3.1 (Path Independent State Mapping): Consider some state s $\in$ smp.s$_0$ $\cup$ smp.SBCA $\cup$ smp.F of a valid SeqMP choreography smp. Then, its mapping to a state s' $\in$ rp.S of some role projection rp of smp created according to the projection algorithm (algorithm objects 4 and

5) does not depend on the path on which s is discovered. □

Proof 3.2:
When creating a projection rp of SeqMP choreography smp for role r, the following types of transitions have to be considered. Thereby, note that upon processing transitions of cases 2-7, the source state already has been mapped.
Case 1: t($smp.s_0$,"true",bca), bca ∈ smp.SBCA
There is no path that starts before $smp.s_0$, so the claim holds for $smp.s_0$ and bca.
Case 2: t(bca1,t#2,bca2), bca1,bca2 ∈ smp.SBCA ∧ r ∈ RA(bca1) ∧ r ∈ RA(bca2)
If bca2 has not been mapped before, then a copy of bca2 is created. Otherwise, its copy is retrieved (cf. algorithm objects 4 and 5). So, the mapping of bca2 and of t is unique.
Case 3: t(bca1,t#2,f), bca1 ∈ smp.SBCA ∧ f ∈ smp.F ∧ r ∈ RA(bca1)
Analogous to case 2.
Case 4: t(bca1,t#2,bca2), bca1,bca2 ∈ smp.SBCA ∧ r ∈ RA(bca1) ∧ r ∉ RA(bca2)
Assume bca2 is not reachable via a second path. Then an event-based choice is created for bca2. Otherwise, if bca2 has not yet been mapped then a dedicated new event-based choice is created as well. If bca2 already has been mapped then by line 22 of algorithm 4 a dedicated new event-based choice has been created for bca2 as well. Transition t then is created between the mapping of bca1 and the event-based choice. So, the mapping of bca2 and t is unique.
Case 5: t(bca1,t#2,bca2), bca1,bca2 ∈ smp.SBCA ∧ r ∉ RA(bca1) ∧ r ∈ RA(bca2)
If bca2 has not been mapped before, then a copy of bca2 is created. Otherwise, its copy is retrieved and a transition between the event-based choice for abstracting bca1 and the copy of bca2 with t#2 as guard is created. So, the mapping of bca2 and t is unique.
Case 6: t(bca1,t#2,f), bca1 ∈ smp.SBCA ∧ f ∈ smp.F ∧ r ∉ RA(bca1)
Analogous to case 5.
Case 7: t(bca1,t#2,bca2), bca1,bca2 ∈ smp.SBCA ∧ r ∉ RA(bca1) ∧ r ∉ RA(bca2)
Assume, bca2 is not reachable via a second path. Then bca2 is mapped to the event-based choice used for abstracting bca1, t is not mapped, and the mapping is unique. Otherwise, if bca2 has not yet been mapped then by line 22 of algorithm 4 a dedicated new event-based choice is created. If it already has been mapped a dedicated new event-based choice has been created for bca2 by the same argument. So, the mapping of bca2 and t is unique.
□

Lemma 3.2 (Isomorphic Subgraph Mappings):
Consider some state s ∈ $smp.s_0$ ∪ smp.SBCA ∪ smp.F of a valid SeqMP choreography smp. Then the graph that results from mapping the forward reachable subgraph of s during some role projection rp of smp is isomorphic for all paths on which s may be discovered. □

Proof 3.3:
Iteratively apply lemma 3.1 during a forward traversal of the graph. □

Lemma 3.3 (Unique Projection Start Pattern):
Consider multiple projections rp_1,...,rp_n of a valid SeqMP choreography smp for the same role r. Then, all of these projections rp_i start with the initial state linking only to an event-based choice: $rp_i.s_0$ = $smp.s_0$ ∧ (∃ t(s_0,"true",e) ∈ rp_i.T. e ∈ rp_i.EBC ∧ ($\nexists t'$ ∈ rp_i.T. $t' \neq t_1$ ∧ t'#1 = $rp_i.s_0$)).
□

Proof 3.4:
If there are multiple projections rp_1,...,rp_n for role r then r does not participate in the first BCA of smp and hence $smp.s_0$ is connected to an event-based choice as of algorithm 5. Otherwise, there would be only a single projection as every state of a valid SeqMP choreography is reachable via the first BCA. □

While all BCAs without participation of the focal role are abstracted away by means of the projection algorithm, the representation of projections still can be optimized. Figure 8 shows one out of two possible projections for the FSP role of the use case depicted in figure 3. It contains only one BCA, but five EBCs and thirteen final states are included to describe permissible behavior. The following five reduction rules have been identified to simplify projections. Rules 3.1, 3.2 and 3.3 do not affect the set of execution traces in the strict formal sense. Rules 3.4 and 3.5 do affect the set of execution traces because final states are removed from the projections and final states are accounted for in execution traces. However, this can be justified by the following consideration: If an EBC links to multiple final states then the focal role is not aware of which final state is reached at runtime (without notification outside the choreography definition). So, the focal role just has the information that the process may be terminated, but it does not know when and with which result. Consequently, all the focal role needs to know is that the process may be terminated after an event-based choice and hence multiple final states after an EBC can be conflated.
By applying the following reduction rules the projection depicted in figure 8 with five EBCs and thirteen final states can be reduced to the projection depicted in figure 9 with only two EBCs and two final states.

Rule 3.1 (Subsequent Event-Based Choices):
Let ebc_1, ebc_2 be event-based choices of a role projection rp of some role r, T^{inter} = {t ∈ rp.T | (t#1 = ebc_1 ∧ t#3 = ebc_2) ∨ (t#1 = ebc_2 ∧ t#3 =

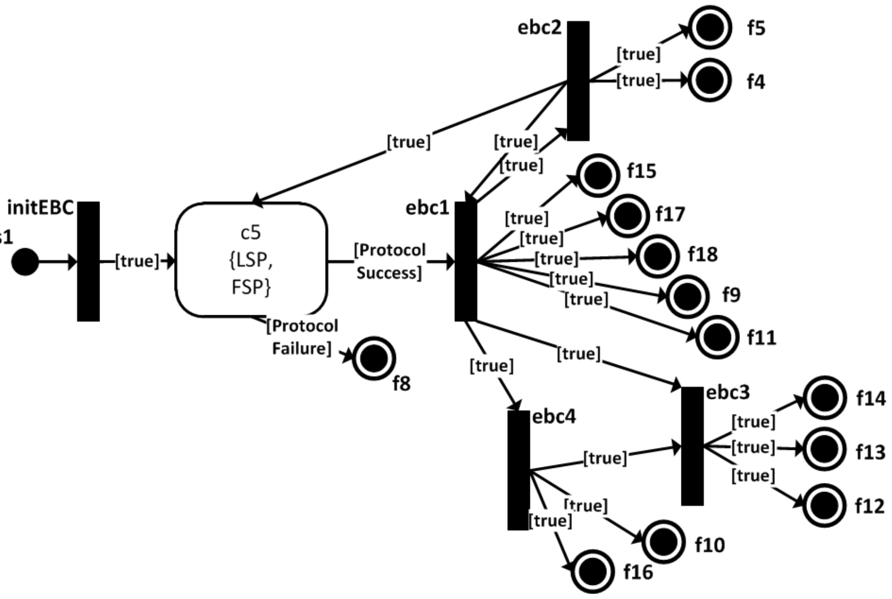

Figure 8. 1 out of 2 Possible Projections for the FSP Role (Use Case of Figure 3)

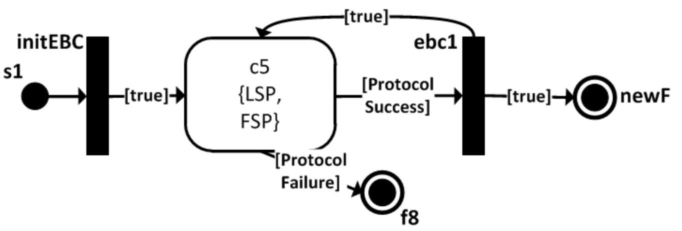

Figure 9. Projection of Figure 8 after Applying Reduction Rules

ebc$_1$)}, T$^{fwd}_{ebc_1}$ = {t ∈ rp.T | t#1 = ebc$_1$} $\setminus$ T^{inter}, T$^{fwd}_{ebc_2}$ = {t ∈ rp.T | t#1 = ebc$_2$} $\setminus$ T^{inter} such that ∄ t ∈ rp.T $\setminus$ T^{inter}. t#3 = ebc$_2$. Then, rp can be simplified without affecting the execution traces of r by performing the following modifications in order:

1) rp.T = rp.T $\setminus$ T^{inter}
2) ∀ t ∈ T$^{fwd}_{ebc_2}$:
   rp.T = rp.T ∪ {(ebc$_1$,t#2,t#3)}
3) rp.T = rp.T $\setminus$ T$^{fwd}_{ebc_2}$
4) rp.EBC = rp.EBC $\setminus$ ebc$_2$

This reduction rule can be applied to conflate ebc1 and ebc2 of figure 8. The informal argument for its correctness is that firing transitions between ebc1 and ebc2 is transparent to the user.

Rule 3.2 (Multiple Event-Based Choices):
Let MEBC = ebc$_1$,...,ebc$_n$ be a set of event-based choices of a role projection rp of some role r indexed by I = [1;n], T^{inter} = {t ∈ rp.T | t#1 = ebc$_i$ ∧ t#3 = ebc$_j$ ∧ i,j ∈ I ∧ i≠j}, T$^{fwd}_{ebc_i}$ = {t ∈ rp.T | t#1 = ebc$_i$} $\setminus$ T^{inter} for i ∈ I such that (∀ i ∈ [2;n]. ∃ t ∈ T^{inter}. t#1 = ebc$_1$ ∧ t#3 = ebc$_i$) ∧ (∀ i ∈ [2;n]. ∄ t ∈ rp.T $\setminus$ T^{inter}. t#3 = ebc$_i$). Then, rp can be simplified without affecting the execution traces of r by performing the following modifications in order:

1) rp.T = rp.T $\setminus$ T^{inter}
2) ∀ t ∈ $\bigcup_{i\in[2;n]}$ T$^{fwd}_{ebc_i}$:
   rp.T = rp.T ∪ {(ebc$_1$,t#2,t#3)}
3) rp.T = rp.T $\setminus$ $\bigcup_{i\in[2;n]}$ T$^{fwd}_{ebc_i}$
4) rp.EBC = rp.EBC $\setminus$ (MEBC $\setminus$ {ebc$_1$})

This reduction rule can be applied to conflate ebc1, ebc3 and ebc4 of figure 8 and the argument for its correctness is analogous to rule 3.1. Strictly speaking this rule is the generalized form of rule 3.1.

Rule 3.3 (Loop):
Let bca be a BCA and MEBC = ebc$_1$,...,ebc$_n$ be a set of event-based choices of a role projection rp of some role r indexed by I = [1;n], PRED$_{bca}$ = {t ∈ rp.T | t#3 = bca}, T^{inter} = {t ∈ rp.T | t#1 = s$_1$ ∧ t#3 = s$_2$ ∧ s$_1$, s$_2$ ∈ MEBC ∪ {bca} ∧ s$_1$ ≠ s$_2$}, T$^{fwd}_{ebc_i}$ = {t ∈ rp.T | t#1 = ebc$_i$} $\setminus$ T^{inter} for i ∈ I such that (∀ i ∈ [2;n]. ∃ t ∈ T^{inter}. t#1 = ebc$_1$ ∧ t#3 = ebc$_i$) ∧ (∀ i ∈ [2;n]. ∃ t ∈ T^{inter}. t#1 = ebc$_i$ ∧ t#3 = ebc$_1$) ∧ (∀ i ∈ [2;n]. ∃ t ∈ T^{inter}. t#1 = bca ∧ t#3 = ebc$_i$) ∧ (∃ t ∈ rp.T. t#1 = ebc$_1$ ∧ t#3 = bca) ∧ (∀ i ∈ [2;n]. ∄ t ∈ rp.T $\setminus$ T^{inter}. t#3 = ebc$_i$). Then, rp can be simplified without affecting the execution traces of r by performing the following modifications in order:

1) rp.T = rp.T $\setminus$ ((T^{inter} $\setminus$ PRED$_{bca}$) $\setminus$ SUCC$_{bca}$)
2) ∀ t ∈ $\bigcup_{i\in[2;n]}$ T$^{fwd}_{ebc_i}$:

rp.T = rp.T $\cup$ {(ebc$_1$,t#2,t#3)}

3) $\forall$ t $\in$ (SUCC$_{bca}$ $\cap$ T^{inter})
   rp.T = rp.T $\cup$ {(t#1,t#2,ebc$_1$)}
4) rp.T = rp.T $\setminus \bigcup_{i\in[2;n]}$ T$^{fwd}_{ebc_i}$
5) rp.T = rp.T $\setminus$ (SUCC$_{bca}$ $\cap$ (T^{inter} $\setminus$ {t $\in$ T^{inter} | t#3 = ebc$_1$}))
6) rp.EBC = rp.EBC $\setminus$ (MEBC $\setminus$ {ebc$_1$})

This rule is not reflected in the projections of figures 6, 7, 8.

Rule 3.4 (Multiple Final States):
Let ebc be an event-based choice of a role projection rp of some role r, T^f a set of transitions such that $\forall$ t $\in$ T^f. t#1 = ebc $\wedge$ t#3 $\in$ rp.F $\wedge$ ($\nexists$ t$'$ $\in$ rp.T. t$'$ $\neq$ t $\wedge$ t$'$#3 = t#3, and T^{alt} a non-empty set of transitions such that $\forall$ t $\in$ T^{alt}. t#1 = ebc $\wedge$ t#3 $\notin$ rp.F. Then, rp can be simplified by performing the following modifications in order:

1) rp.T = rp.T $\setminus$ T^f
2) rp.F = rp.F $\setminus$ {f| $\exists$ t $\in$ T^f. t#3 = f}
3) rp.F = rp.F $\cup$ f, f a new FinalState
4) rp.T = rp.T $\cup$ {(ebc,"true",f)}

This reduction rule can be applied to conflate all final states of figure 8 connected to ebc1, ebc2, ebc3 and ebc4 after having applied rules 3.2 and 3.1. The rationale for doing so has been explained above.

Rule 3.5 (Event-Based Choice and Final States):
Let ebc be an event-based choice of a role projection rp of some role r, PRED$_{ebc}$ = {t $\in$ rp.T | t#3 = ebc}, T^{final} = {t $\in$ rp.T | t#1 = ebc $\wedge$ t#3 $\in$ rp.F} such that $\nexists$ t$'$ in rp.T $\setminus$ T^{final}. t$'$#1 = ebc $\vee$ t$'$#3 = t#3 for some t $\in$ T^{final}. Then, rp can be simplified by performing the following modifications in order:

1) rp.T = rp.T $\setminus$ T^{final}
2) rp.F = rp.F $\setminus$ {s| $\exists$ t $\in$ T^{final}. t#3 = s}
3) rp.F = rp.F $\cup$ f, f a new FinalState
4) $\forall$ t $\in$ PRED$_{ebc}$:
   rp.T = rp.T $\cup$ {(t#1,t#2,f)},
   rp.T = rp.T $\setminus$ {t}
5) rp.EBC = rp.EBC $\setminus$ {ebc}

This rule can be applied to conflate ebc1, f12, f13, and f14 of figure 7.

IV. Related Work

Generally speaking, the work at hand belongs to the domain of business process management. In this domain, research on developing implementations or analyzing artifacts at the implementation level as presented in [15], [16] or [17] is very common. However, our work is substantially different. We provide a framework for analyzing multi-party choreographies at an abstract level. This work neither strives for automatically solving the partial termination problem nor for deriving implementations of the choreographies or of choreography projections.

The problems identified in section III-B are quite different from several problems identified in different choreography research. In reports such as [18], [19], [20], [21], problems like *enforceability* or *realizability* are researched. The according approaches have in common that the atomic building blocks of choreographies are single message exchanges and it is then researched whether or not the message sequences in the choreographies can be *enforced* by the local role-specific projections of interaction partners, and whether or not the sequence of message exchanges is the same for synchronous or asynchronous communication. In the work at hand, these problems are not relevant. By using BCAs instead of single message exchanges, we can be sure that the state of integration partners is aligned at the end of each BCA (cf. [4], [13]). BCAs are performed using according protocol machines that ensure alignment and therefore are not comparable to single messages in some communication buffer that may cause diverging states or deadlocks. Moreover, due to the *Subsequent role participation* condition, valid SeqMPs do not suffer from a local enforceability problem. To the best of our knowledge, *partial termination* as defined above has first been identified as multi-party choreography problem in [6].

However, deriving role projections of multi-party interactions has been a research topic for a long time. For example, van der Aalst and Weske [22] describe an approach for dissecting Petri Nets that contain role specific behavior. However, communication between participants is modeled as *send*- and *receive*-transitions that already are associated with roles. This corresponds to dissecting interconnection style choreographies and is representative for more recent research on interconnection style choreographies. However, we are considering the problem of deriving role projections from interaction style choreographies. At first sight, the proposal for deriving role projections of *Let's Dance* models as described in [18] is comparable to that problem. However, Let's Dance applies a block-structured approach for modeling loops which is different from the class of SeqMP models with arbitrary loops. In [23], so-called Interaction Petri nets that enable multi-party interactions by modeling binary message exchanges as transitions of Petri nets, are analyzed. The model of [23] therefore can be considered to be comparable for the role projection problem of the work at hand and an algorithm for calculating role projections is presented as well. However, the problem there is defined for Petri Nets and therefore the algorithm is not directly amenable to SeqMP models. Further, the algorithm in [23] introduces duplicate transitions for flattening parallelism. However, transitions correspond to message exchanges in [23] and to BCAs for SeqMP. The business semantics of duplicate message exchanges (or duplicate

BCAs for SeqMP) is not clear, though.
Apart from that, interesting choreography work has been presented in [5] where so-called local choreographies are used to model the sequence of interactions of one integration partner in multiple global choreographies. The perspective is different from the work at hand by focusing on a partner that participates in more than one choreography where the participants of the respective choreographies may only know the focal integration partner. A typical scenario for that type of integration is a manufacturer that employs a sub-contractor producing parts of a product without the customer knowing the sub-contractor. While there may be valid reasons for not revealing the interactions with business partners to different business partners, the type of integration in [5] leaves out the opportunity to perform analyses like escalation set computation that rely on a global view on choreographies.

V. Conclusion and Future Work

This paper introduces an approach for creating multi-party B2Bi choreographies from binary B2Bi choreographies. The identification of the *partial termination* and the *role projection* problem show the potential of SeqMP for optimizing multi-party B2Bi scenarios. We have provided an algorithm that allows for analyzing the partial termination problem using three distinct strategies that trade off effort for the user and configurability and we have provided an algorithm for creating role specific projections. Both algorithms have been validated using a prototype implementation. Additionally, we have shown how to merge multiple projections of the same role such that the individual projections and their merge are behaviorally equivalent and we have identified rules for reducing projections.
Future work comprises defining an integration architecture for supporting the communication of *partial termination* events among the participants of a SeqMP choreography. Moreover, the derivation and monitoring of legally binding obligations to communicate *partial termination* events is still an open issue.

References

[1] J.-H. Kim and C. Huemer, "From an ebXML BPSS choreography to a BPEL-based implementation," *SIGecom Exch.*, vol. 5, no. 2, pp. 1–11, 2004.

[2] A. Schönberger and G. Wirtz, "Using Webservice Choreography and Orchestration Perspectives to Model and Evaluate B2B Interactions," in *The 2006 International Conference on Software Engineering Research and Practice (SERP'06), Las Vegas, Nevada, USA*, June 26-29 2006, pp. 329–335.

[3] S. Wieczorek, A. Roth, A. Stefanescu, V. Kozyura, A. Charfi, F. M. Kraft, and I. Schieferdecker, "Viewpoints for modeling choreographies in service-oriented architectures," in *Joint Working IEEE/IFIP Conference on Software Architecture 2009 & European Conference on Software Architecture 2009 (WICSA/ECSA), Cambridge. Los Alamitos/Calif.*, 2009.

[4] A. Schönberger and G. Wirtz, "Towards executing ebBP-Reg B2Bi choreographies," in *Proceedings of the 12th IEEE Conference on Commerce and Enterprise Computing (CEC'10), Shanghai, China*. IEEE, November 10-12 2010.

[5] B. Hofreiter and C. Huemer, "A model-driven top-down approach to inter-organizational systems: From global choreography models to executable BPEL," in *Joint Conference on E-Commerce Technology (CEC'08) and Enterprise Computing, E-Commerce, and E-Services (EEE'08)*. Crystal City, Washington D.C., USA: IEEE, 7 2008.

[6] A. Schönberger and G. Wirtz, "Sequential composition of multi-party choreographies," in *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA'10), Perth, Australia*. IEEE, December 13-15 2010.

[7] OASIS, *ebXML Business Process Specification Schema Technical Specification*, 2nd ed., OASIS, December 2006. [Online]. Available: http://docs.oasis-open.org/ebxml-bp/2.0.4/OS/spec/ebxmlbp-v2.0.4-Spec-os-en.pdf

[8] G. Decker, O. Kopp, and A. Barros, "An introduction to service choreographies," *Information Technology*, vol. 50, no. 2, pp. 122–127, 2008.

[9] UN/CEFACT, *UN/CEFACT's Modeling Methodology (UMM): UMM Meta Model - Foundation Module Version 1.0*, 1st ed., UN/CEFACT, 10 2006.

[10] M. Zapletal, T. Motal, and H. Werthner, "The business choreography language (BCL) - a domain-specific language for global choreographies," in *Proceedings of the 5th 2009 World Congress on Services (SERVICES 2009 PART II), International Workshop on Services Computing for B2B (SC4B2B), Bangalore, India*. IEEE, September 2009.

[11] OMG, *Business Process Model and Notation, v2.0*, OMG, January 2011. [Online]. Available: http://www.omg.org/spec/BPMN/2.0

[12] A. Schönberger, C. Pflügler, and G. Wirtz, "Translating shared state based ebXML BPSS models to WS-BPEL," *International Journal of Business Intelligence and Data Mining - Special Issue: 11th International Conference on Information Integration and Web-Based Applications and Services in December 2009*, vol. 5, no. 4, pp. 398 – 442, 2010.

[13] C. Pflügler, A. Schönberger, and G. Wirtz, "Introducing partner shared states into ebBP to WS-BPEL translations," in *Proc. iiWAS2009, 11th International Conference on Information Integration and Web-based Applications & Services, 14.-16. December 2009, Kuala Lumpur, Malaysia*. ACM, December 2009.

[14] H. A. Reijers and W. M. van der Aalst, “The effectiveness of workflow management systems: Predictions and lessons learned,” *International Journal of Information Management*, vol. 25, no. 5, pp. 458 – 472, 2005.

[15] N. Laranjeiro and M. Vieira, “Deploying fault tolerant web service compositions,” *International Journal of Computer Systems Science and Engineering, Special Issue:âEngineering Fault Tolerant Systems*, vol. 23, no. 5, pp. 337–348, Sep 2008.

[16] E. F.-M. Alfonso Rodríguez and M. Piattini, “An MDA approach to develop secure business processes through a UML 2.0 extension,” *International Journal of Computer Systems Science and Engineering, Special Issue:âTrustBus 2006*, vol. 22, no. 5, pp. 307–319, Sep 2007.

[17] M. Kovács, D. Varró, and L. Gönczy, “Formal analysis of BPEL workflows with compensation by model checking,” *International Journal of Computer Systems Science and Engineering, Special Issue: Engineering Fault Tolerant Systems*, vol. 23, no. 5, pp. 349–363, Sep 2008.

[18] J. Zaha, M. Dumas, A. ter Hofstede, A. Barros, and G. Decker, “Bridging global and local models of service-oriented systems,” *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 38, no. 3, pp. 302 – 318, may 2008.

[19] G. Decker, A. Barros, F. M. Kraft, and N. Lohmann, “Non-desynchronizable service choreographies,” in *ICSOC ’08: Proceedings of the 6th International Conference on Service-Oriented Computing*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 331–346.

[20] T. Bultan, J. Su, and X. Fu, “Analyzing conversations of web services,” *IEEE Internet Computing*, vol. 10, no. 1, pp. 18–25, 2006.

[21] V. Kozyura, A. Roth, and W. Wei, “Local enforceability and inconsumable messages in choreography models,” in *Proceedings of 4th South-East European Workshop on Formal Methods (SEEFM’09),Thessaloniki, Greece*, 2009.

[22] W. M. P. van der Aalst and M. Weske, “The P2P approach to interorganizational workflows,” in *CAiSE ’01: Proceedings of the 13th International Conference on Advanced Information Systems Engineering*. London, UK: Springer-Verlag, 2001, pp. 140–156.

[23] G. Decker and M. Weske, “Local enforceability in interaction petri nets,” in *Proceedings of the 5th International Conference on Business Process Management (BPM 2007), Brisbane, Australia, September 24-28*, 2007, pp. 305–319.