Exploring the Connection of Choreography and Orchestration with Exception Handling and Finalization/Compensation [⋆]

Yang Hongli, Zhao Xiangpeng, Cai Chao, and Qiu Zongyan

LMAM and Department of Informatics, School of Math.,
Peking University, Beijing 100871, China
{yhl,zxp,caic,qzy}@math.pku.edu.cn

Abstract. Web service choreography describes protocols for multiparty collaboration, whereas orchestration focuses on single peers. One key requirement of choreography is to support transactions, which makes exceptional handling and finalization very important features in modelling choreography. A *projection* is a procedure which takes a choreography and generates a set of processes in the orchestration level. Given a choreography, how to project exceptional handling and finalization constructs is still an open problem. This paper aims to study exception handling and transactionality in choreographies from a projection view. We propose formal languages for both choreography and orchestration with trace semantics, and a projection based on the relationship between *choreography* and *scope* rooted in WS-CDL and WS-BPEL respectively.

Keywords: Choreography, Orchestration, Projection, Exception Handling, Finalization, Compensation

1 Introduction

Web services promise the interoperability of various applications running on heterogeneous platforms over the Internet. Web service composition refers to the process of combining web services to provide value-added services, which has received much interest to support enterprise application integration.

Two levels of view to the composition of web services exist, namely orchestration and choreography. The description of the single services, possibly with cooperation of other services, is called an orchestration. The *de facto* standard for orchestration is WS-BPEL [3] (Web Services Business Process Execution Language) developed by a consortium comprising BEA, IBM, Microsoft etc. The global view of the interactions are described by the so-called choreography. WS-CDL(Web Service Choreography Description Language) [2] is a W3C candidate recommendation, designed for describing the common and collaborative observable behavior of multiple services that interact with each other. Another

[⋆] Supported by National Natural Science Foundation of China (No. 60573081) notation, SSDL [1], also allows the description of protocols for multiparty collaboration using message-oriented programming abstractions. In short, choreography describes the system in a global-view manner whereas orchestration focuses on the peers separately.

Using WS-CDL, a contract contains a "global" definition of the common flow ordering conditions and constraints of a task, which should be in turn realized by combination of the several local systems [2]. Once the contract is clearly defined and jointly agreed to, participants can be built and tested according to it independently. However, two challenges exist: (1) how to automatically generate correct local requirements for the roles from the global contract; (2) how to verify whether a given process can play as a participant whose observable behavior conforms to the requirement of a given choreography.

Much work has been carried out, while much is still going on in the projection and conformance validation between choreography and orchestration. Carbone *et al.* [11] studied a two-level paradigm for the description of communication behaviors, on the global message flows and end-point behavior levels respectively. Three principles for well-structured global description and a theory for projection are developed. In [7, 8], Busi *et al.* formalized choreography and orchestration by using process algebra, where conformance takes the form of a bisimulation-like relation. By means of automaton, Schifanella *et al.* [4] defined a conformance notion which tests whether interoperability is guaranteed. Fu *et al.* [12] specified a conversation protocol by a realizable Büchi automaton, and the peer implementations are synthesized from the protocol via projection. Zhao *et al.* [17] proposed a small language as a formal model of the simplified WS-CDL and projected a given choreography to orchestration views.

One key aspect in composing web services is to support transactions of process executions. Exception handling and transactionality are important features in both choreography and orchestration levels. WS-CDL provides finalizer actions to confirm, cancel or modify the effects of its completed actions. In orchestration level, if a long-running transaction fails, appropriate compensations are executed for the completed parts of the transaction, which is supported by WS-BPEL with its scope-based compensation. Butler *et al.* integrated the compensation feature into CSP, and provided both operational semantics and denotational (trace) semantics [9, 10]. Bruni *et al.* presented a hierarchy of transactional calculi with increasing expressiveness in [6]. Qiu *et al.* [15] and Pu *et al.* [14] studied the semantics of WS-BPEL fault and compensation handling. Li *et al.* [13] proposed a language with operational semantics to model exception handling and finalization of WS-CDL. To the best of our knowledge, no work is done about modelling exception handling and transactionality from a projection view, which resolves how exception handling and finalization in a choreography can be implemented in orchestration level.

In our previous work [16], we have presented a simplified language for choreography, and a simple process language for participant roles, both with formal syntax and semantics. We discussed the concept of projections, which map a given choreography to a set of role processes. We defined the concept of *re-*

stricted natural choreography which is easily implementable, and proposed two structural conditions as a criterion to distinguish the restricted natural choreography. Although useful as a formal investigation of the relationship between choreography and orchestration, the framework is not powerful enough to specify real case studies. The main weak point for the expressiveness is the shortage of mechanism for describing exception handling and transactionality.

This paper aims at extending our framework for both choreography and orchestration with structures related to exception handling and transactionality. The choreography language *Chor* and orchestration language *Role*, which are inspired by WS-CDL and WS-BPEL respectively, are developed with formal syntax and trace semantics. We present a projection from *Chor* to *Role* which focuses on the relationship between *choreography* in *Chor* and *scope* in *Role* rooted in WS-CDL and WS-BPEL respectively. Both the two structures have actions, exception block, finalizer or compensation action. Because of their similarities, our projection will map a *choreography* in *Chor* to a *scope* at each role process in *Role*. In the work, we develop a technique for define trace semantics of the role process language that introduces a stuck notation.

The rest of the paper is organized as follows. We first introduce the syntax and semantics of *Chor* with exception handling in Section 2. Then we add the finalization feature into *Chor* language in Section 3. Section 4 defines a *Role* language with formal syntax and semantics. Section 5 presents the projection rules with some discussion about the related issue, and Section 6 concludes.

2 The *Chor* Language with Exception Handling

In this section we develop the language *Chor* with syntax and trace semantics.

2.1 Syntax

In the definitions below, A and B range over activity declarations; E ranges over exception blocks; e ranges over exceptions; and n ranges over names. We use $\overline{X}$ as a shorthand for list, similarly, for $\overline{e : A}$. Given a list l, $\mathsf{hd}(l)$ returns the first element of l, and $\mathsf{tl}(l)$ returns the same list with the first element removed.

A choreography is participated by a finite number of roles $\overline{R^1, \cdots, R^n}$. A choreography specification comprises some choreography declarations $\overline{CDecl}$ and a root choreography RC.

$$CS \ ::= \ \overline{CDecl}, RC$$

The root choreography is enabled by default, whereas other choreography are enabled only when they are performed. The root choreography is a tuple, including an activity A, and an exception block E.

$$RC \ ::= \ [A, E]$$

A declaration of a non-root choreography with name n takes the form:

$$CDecl \ ::= \ n[A, E]$$

Here is the syntax for the activities in *Chor*.

$$\begin{array}{rlll} A ::= \mathsf{skip} & \text{(no action)} & \mid a^i & \text{(activity)} \\ \mid\ c^{[i,j]} & \text{(communication)} & \mid \mathsf{throw}\ e & \text{(throw exception)} \\ \mid\ \mathsf{perf}\ n & \text{(perform)} & \mid A; A & \text{(sequence)} \\ \mid\ A \overset{i}{\sqcap} A & \text{(choice)} & \mid A \parallel A & \text{(parallel)} \end{array}$$

Activity skip does nothing. Meta-variable a^i denotes a basic activity of role R^i. The communication from R^i to R^j takes the form of $c^{[i,j]}$, where c is a channel name. Activity throw e causes an exception e at each role. Activity perf n performs the declared choreography with name n. The composite activities considered here include sequential composition, choice, and parallel composition. Here $A \overset{i}{\sqcap} A$ means that role R^i is the *dominant role* of the choice. It is used as a directive in projection to specify that R^i is the "decision maker", and all other roles should follow R^i's decision on which branch to take in this choice. A more detailed study about the dominant role can be found in [16].

The exception block E is defined as a sequence of $e : A$, where e is an exception name, and the activity A is the exception handler for e. We allow $* : A$ as a special case to define a universal handler in an exception block.

$$E \ ::= \ \overline{e : A}$$

A choreography specification is well-formed if all the following conditions hold:

– All non-root choreography names are different from each other.
– In each perform activity perf n, the name n ranges over non-root choreography names in the choreography specification.
– All exception names in each exception block are different from each other.

2.2 Semantics

An environment Γ is a map from non-root choreography names to their definitions with the form $[A, E]$, which can be constructed by parsing the text of declarations $\overline{CDecl}$. We will assume that the execution of a choreography is always under the corresponding Γ. For convenience, notation $n.1, n.2$ will be used to obtain the activity and the exception block of choreography n.

We define the semantics of an activity as a set of traces, and will use r, s, and t to denote traces. A trace may have a terminal mark at its end, indicating whether the execution of the activities terminates successfully or not. Mark $\checkmark$ represents a successful termination, and $\Lsh_e$ represents a termination with exception e. Concatenation of traces is denoted by juxtaposition. For example, $t\langle\checkmark\rangle$ represents a concatenated trace which terminates successfully. In our semantics, we always give maximal traces, i.e. each trace has a terminal mark at its end.

Activity skip does nothing and always terminates successfully. Activity throw e causes exception e. Activity a^i always terminates successfully, so does $c^{[i,j]}$.

$$\begin{array}{rclrcl} [\![\mathsf{skip}]\!]_\Gamma & \widehat{=} & \{\langle\checkmark\rangle\} & [\![a^i]\!]_\Gamma & \widehat{=} & \{\langle a^i, \checkmark\rangle\} \\ [\![\mathsf{throw}\ e]\!]_\Gamma & \widehat{=} & \{\langle\Lsh_e\rangle\} & [\![c^{[i,j]}]\!]_\Gamma & \widehat{=} & \{\langle c^{[i,j]}, \checkmark\rangle\} \end{array}$$

To define the semantics of the perform activity $\mathsf{perf}\ n$, we need to define the semantics of executing an exception block under some exception e. We introduce function $hdl(E,e)_\Gamma$, which returns a set of traces after handling exception e in exception block E under environment Γ. If a handler for e, which may take the form of $e : A$ or $* : A$, is found in E, then the traces of A are returned. Otherwise, the exception will be propagated to the immediate enclosing choreography.

$$hdl(E,e)_\Gamma \mathrel{\widehat{=}} \begin{cases} [\![A]\!]_\Gamma & \text{if } \mathsf{hd}(E) = e : A \vee \mathsf{hd}(E) = * : A \\ hdl(\mathsf{tl}(E), e)_\Gamma & \text{if } \mathsf{hd}(E) = e' : A \wedge e' \neq e \\ \{\langle \Rsh_e \rangle\} & \text{if } E \text{ is empty} \end{cases}$$

Now we define the semantics of the perform activity as:

$$[\![\mathsf{perf}\ n]\!]_\Gamma \mathrel{\widehat{=}} \{s\langle\checkmark\rangle \mid s\langle\checkmark\rangle \in [\![n.1]\!]_\Gamma\} \ \cup\ \{st \mid s\langle\Rsh_e\rangle \in [\![n.1]\!]_\Gamma \wedge t \in hdl(n.2, e)_\Gamma\}$$

If activity $n.1$ terminates successfully, so is the perform activity. Otherwise, if $n.1$ throws an exception e, the exception handler in $n.2$ for e is executed, and the trace t produced by this execution is appended to trace s.

The semantics of choice is defined by set union. Although i does not appear in the semantics, it is critical in the projection discussed in Section 5.

$$[\![A \overset{i}{\sqcap} B]\!]_\Gamma \ \mathrel{\widehat{=}}\ [\![A]\!]_\Gamma \cup [\![B]\!]_\Gamma$$

We introduce the sequential and parallel composition of traces, and lift them to sets of traces, then give semantics of sequential and parallel composition.

$$\begin{aligned} s\langle\checkmark\rangle; t &\mathrel{\widehat{=}} st \qquad\qquad s\langle\Rsh_e\rangle; t \ \mathrel{\widehat{=}}\ s\langle\Rsh_e\rangle \\ [\![A;B]\!]_\Gamma &\mathrel{\widehat{=}} \{s;t \mid s \in [\![A]\!]_\Gamma \wedge t \in [\![B]\!]_\Gamma\} \\ s\langle\tau\rangle \parallel t\langle\tau'\rangle &\mathrel{\widehat{=}} \{r\langle\tau \oplus \tau'\rangle \mid r \in interl(s,t)\} \\ [\![A \parallel B]\!]_\Gamma &\mathrel{\widehat{=}} \{r \mid r \in (s \parallel t) \wedge s \in [\![A]\!]_\Gamma \wedge t \in [\![B]\!]_\Gamma\} \end{aligned}$$

Here τ and τ' are meta variables over terminal marks $\{\checkmark, \Rsh_e\}$. The function $interl(s,t)$ denotes the set of all interleaving traces of s and t. The definition is routine and is omitted here. The terminal mark of parallel composition is defined by operator $\oplus$, as shown in the table below.

$\oplus$	$\checkmark$	$\Rsh_{e_1}$
$\checkmark$	$\checkmark$	$\Rsh_{e_1}$
$\Rsh_{e_2}$	$\Rsh_{e_2}$	$\Rsh_{e_1 \uplus e_2}$

If both branches terminate successfully, so is their parallel composition. When both branches terminate with some exception(s), then we need to handle the parallel exceptions by operator $\uplus$. There are many possible ways to define $\uplus$. For example, we can define different priorities for exceptions and return the highest one; or define a hierarchy of exceptions and return the least upper bound. The details of handling parallel exceptions are omitted here. If only one branch fails, we have the exception for the parallel composition. We do not consider the forced termination problem [3] in this paper.

Provided the semantics of activities, we can define the semantics of the root choreography as follows, which is similar with the perform activity:

$$[\![[A, E]]\!]_\Gamma \mathrel{\widehat{=}} \{s\langle\checkmark\rangle \mid s\langle\checkmark\rangle \in [\![A]\!]_\Gamma\} \;\cup\; \{st \mid s\langle\Rsh_e\rangle \in [\![A]\!]_\Gamma \wedge t \in hdl(E, e)_\Gamma\}$$

Many laws for structural congruence, e.g., associativity and symmetry, hold for choice and parallel composition. Also, skip is the unit element of the sequential operator, and throw e the left zero, i.e. throw $e; A =$ throw e. Besides, we can easily prove that any choreography will always terminate, either successes or fails with an exception, i.e. any choreography is deadlock-free.

Now we present an example to illustrate the semantics.

Example 1. In the following declaration, notation a_l^1, a_m^1 and a_n^1 denote basic activities at role R^1.

$$m[(a_l^1; \mathsf{throw}\ e_n), e_m : a_m^1], \quad [\mathsf{perf}\ m, e_n : a_n^1]$$

Here environment Γ consists of a map from choreography name m to its body, i.e. $\Gamma = \{m \mapsto [(a_l^1; \mathsf{throw}\ e_n), e_m : a_m^1]\}$. When the root choreography $[\mathsf{perf}\ m, e_n : a_n^1]$ executes under Γ, choreography m is performed. The activity a_l^1 is executed first, and then exception e_n is thrown. Since e_n cannot be handled in m, it is re-thrown to the root choreography, where e_n is handled by the exception block. When activity a_n^1 terminates successfully, the root choreography terminates. Thus, we derive the following semantics:

$$[\![\mathsf{perf}\ m]\!]_\Gamma = \{\langle a_l^1, \Rsh_{e_n}\rangle\} \qquad [\![[\mathsf{perf}\ m, e_n : a_n^1]]\!]_\Gamma = \{\langle a_l^1, a_n^1, \checkmark\rangle\}$$

3 Adding Finalization

In this section, we extend *Chor* language with constructs for finalization.

The non-root choreography declaration is extended to include a finalizer F, with the form:

$$CDecl \ ::= \ n[A, E, F] \qquad F \ ::= \ A$$

Unlike the case for exceptions, we do not consider named finalizers, and F is simply defined as an activity for finalization. However, there is no substantial difficulty to extend the model to support named finalization.

The syntax of activities is extended with the finalize activity fin n, which performs the finalizer of the successfully terminated choreography n.

$$A \ ::= \ \cdots \ \mid \ \mathsf{fin}\ n \ \mid \ \cdots$$

After introducing finalization structures, we need to extend the semantics, since finalizers are dynamically installed during the execution of choreographies. If the performing of a choreography n terminates successfully, the finalizer of n will be installed.

In the definitions below, meta-variable φ, ψ, χ range over finalization contexts. A finalization context is a (possibly empty) sequence of finalization closures of the form $(n : F : \psi)$, where n is a choreography name, F the finalizer of n, and ψ the finalization context accumulated during performing choreography n, as n might perform some other choreographies in its course. Here is an example of a finalization context: $\langle (n_1 : F_1 : \langle\rangle), (n_2 : F_2 : \langle (n_3 : F_3 : \langle\rangle)\rangle)\rangle$.

We express the semantics of an activity as a set of pairs with the form (s, φ'), where s represents a trace of the activity, and φ' represents the new finalization context after executing the activity. We always assume that the execution of activity is under some finalization context φ and environment Γ (now with elements of the form $n \mapsto [A, E, F]$). Initially, φ is empty.

The basic activities skip, a^i, $c^{[i,j]}$ and $\mathsf{throw}\ e$ have no effect on the finalization context, so the extension is trivial.

$$\begin{array}{ll}
\llbracket \mathsf{skip} \rrbracket_\Gamma^\varphi \mathrel{\widehat{=}} \{(\langle \checkmark \rangle, \varphi)\} & \llbracket \mathsf{throw}\ e \rrbracket_\Gamma^\varphi \mathrel{\widehat{=}} \{(\langle \Rsh_e \rangle, \varphi)\} \\
\llbracket a^i \rrbracket_\Gamma^\varphi \mathrel{\widehat{=}} \{(\langle a^i, \checkmark \rangle, \varphi)\} & \llbracket c^{[i,j]} \rrbracket_\Gamma^\varphi \mathrel{\widehat{=}} \{(\langle c^{[i,j]}, \checkmark \rangle, \varphi)\}
\end{array}$$

For the perform activity $\mathsf{perf}\ n$, if activity $n.1$ completes successfully, closure $(n : n.3 : \psi)$ is inserted in front of φ, where $n.3$ is the finalizer of choreography n, and ψ is the accumulated finalization context during performing choreography n. If $n.1$ throws an exception, φ remains the same. Symbol "$-$" means something that we do not care about.

$$\begin{array}{rcl}
\llbracket \mathsf{perf}\ n \rrbracket_\Gamma^\varphi & \widehat{=} & \{(s\langle \checkmark \rangle, (n : n.3 : \psi)^\frown \varphi) \mid (s\langle \checkmark \rangle, \psi) \in \llbracket n.1 \rrbracket_\Gamma^{\langle\rangle}\} \ \cup \\
& & \{(st, \varphi) \mid (s\langle \Rsh_e \rangle, \psi) \in \llbracket n.1 \rrbracket_\Gamma^{\langle\rangle} \wedge (t, -) \in hdl(n.2, e)_\Gamma^\psi\}
\end{array}$$

Here $hdl(E, e)_\Gamma^\varphi$ is an extension of $hdl(E, e)_\Gamma$. When E is empty, the exception is rethrown to the performer of current choreography.

$$hdl(E, e)_\Gamma^\varphi \mathrel{\widehat{=}} \begin{cases} \llbracket A \rrbracket_\Gamma^\varphi & \text{if } \mathsf{hd}(E) = e : A \vee \mathsf{hd}(E) = * : A \\ hdl(\mathsf{tl}(E), e)_\Gamma^\varphi & \text{if } \mathsf{hd}(E) = e' : A \wedge e' \neq e \\ \{(\langle \Rsh_e \rangle, -)\} & \text{if } E \text{ is empty} \end{cases}$$

The semantics of $\mathsf{fin}\ n$ is defined as follows. We assume the execution of a finalizer does not modify the current finalization context. Function $getf(n, \varphi)$ gets the finalizer F of choreography n from φ by searching through the context. Similar to the specification of WS-CDL, if no corresponding finalizer found, nothing happens. If closure $(n : F : \psi)$ is found, we execute F under ψ.

$$\llbracket \mathsf{fin}\ n \rrbracket_\Gamma^\varphi \mathrel{\widehat{=}} \{(s, \varphi) \mid (s, -) \in getf(n, \varphi)_\Gamma\}$$

$$getf(n, \varphi)_\Gamma \mathrel{\widehat{=}} \begin{cases} \llbracket \mathsf{skip} \rrbracket_\Gamma^\varphi & \text{if } \varphi = \langle\rangle \\ \llbracket F \rrbracket_\Gamma^\psi & \text{if } \mathsf{hd}(\varphi) = (n : F : \psi) \\ getf(n, \mathsf{tl}(\varphi))_\Gamma & \text{if } \mathsf{hd}(\varphi) = (n' : F : \psi) \wedge n \neq n' \end{cases}$$

For sequential composition, we first execute A under context φ. Suppose the context becomes ψ after the execution; we then execute B under ψ, which results in context χ. If A ends with an exception execution, then B does not execute.

$$\begin{aligned}[\![A;B]\!]_{\Gamma}^{\varphi} \;\widehat{=}\; & \{(st,\chi) \mid (s\langle\checkmark\rangle,\psi) \in [\![A]\!]_{\Gamma}^{\varphi} \wedge (t,\chi) \in [\![B]\!]_{\Gamma}^{\psi}\} \;\cup \\ & \{(s\langle\Rsh_e\rangle,\psi) \mid (s\langle\Rsh_e\rangle,\psi) \in [\![A]\!]_{\Gamma}^{\varphi}\}\end{aligned}$$

For parallel composition, we execute both branches under context φ and environment Γ, and then combine the traces and accumulated finalization closures interleavingly. Here $s \parallel t$ and *interl* have the same meaning as in Section 2.2.

$$\begin{aligned}[\![A \parallel B]\!]_{\Gamma}^{\varphi} \;\widehat{=}\; & \{(r,\chi) \mid r \in (s \parallel t) \;\wedge \chi \in (interl(\varphi',\varphi'')^\frown \varphi) \;\wedge \\ & (s,\varphi'^\frown \varphi) \in [\![A]\!]_{\Gamma}^{\varphi} \;\wedge (t,\varphi''^\frown \varphi) \in [\![B]\!]_{\Gamma}^{\varphi}\}\end{aligned}$$

The semantics of choice activity is simple: $[\![A \overset{i}{\sqcap} B]\!]_{\Gamma}^{\varphi} \;\widehat{=}\; [\![A]\!]_{\Gamma}^{\varphi} \cup [\![B]\!]_{\Gamma}^{\varphi}$.

The semantics for the root choreography is similar to the semantics of the perform activity:

$$\begin{aligned}[\![[A,E]]\!]_{\Gamma} \;\widehat{=}\; & \{s\langle\checkmark\rangle \mid (s\langle\checkmark\rangle,-) \in [\![A]\!]_{\Gamma}^{\langle\rangle}\} \;\cup \\ & \quad \{st \mid (s\langle\Rsh_e\rangle,\psi) \in [\![A]\!]_{\Gamma}^{\langle\rangle} \wedge (t,-) \in hdl(E,e)_{\Gamma}^{\psi}\}\end{aligned}$$

We show the use of the finalizer construct with the following example.

Example 2. This example includes three non-root choreographies m, n and p. Here a_m^1 and a_f^1 denote basic activities at role R^1; a_p^2 and a_f^2 denote basic activities at role R^2. The notation ϵ denotes that the exception block is empty.

$$m[a_m^1,\epsilon,a_f^1] \qquad n[\mathsf{perf}\ m,\epsilon,\mathsf{fin}\ m] \qquad p\,[a_p^2,\epsilon,a_f^2]$$

In the root choreography, choreographies n and p are performed in parallel. Afterwards, exception e is thrown and handled by the root choreography.

$$[((\mathsf{perf}\ n \parallel \mathsf{perf}\ p);\mathsf{throw}\ e), e : (\mathsf{fin}\ n)]$$

Initially, choreographies n and p run in parallel with empty finalization context and environment Γ, which maps the choreography names to bodies of three non-root choreographies. Before $\mathsf{fin}\ n$, the finalization context is $\langle (n : \mathsf{fin}\ m : \langle (m : a_f^1 : \langle\rangle)\rangle),\ (p : a_f^2 : \langle\rangle)\rangle$, or in the reverse order. Then $\mathsf{fin}\ n$ executes $\mathsf{fin}\ m$, which turns to execute activity a_f^1. Afterwards, the root choreography terminates successfully. The two perform activities yield the following traces:

$$\begin{aligned}[\![\mathsf{perf}\ p]\!]_{\Gamma}^{\langle\rangle} &= \{(\langle a_p^2,\checkmark\rangle, \langle (p : a_f^2 : \langle\rangle)\rangle)\} \\ [\![\mathsf{perf}\ n]\!]_{\Gamma}^{\langle\rangle} &= \{(\langle a_m^1,\checkmark\rangle, \langle (n : \mathsf{fin}\ m : \langle (m : a_f^1 : \langle\rangle)\rangle)\rangle)\}\end{aligned}$$

The trace set of the root choreography is $\{\langle a_m^1, a_p^2, a_f^1, \checkmark\rangle, \langle a_p^2, a_m^1, a_f^1, \checkmark\rangle\}$. $\square$

4 The *Role* Language

A choreography describes the interaction among roles from a global view. It is intended to be implemented by coordination of a set of independent processes. In order to study the relationship between the globally described choreography and the coordinative activities of each role, we define a simple *Role* language here. The syntax and the trace semantics are defined as follows.

4.1 Syntax

In the definitions below, P ranges over processes. The syntax of *Role* is:

$$\begin{array}{rlllll}
P ::= & \mathsf{skip} & \text{(no action)} & \mid a & \text{(local action)} \\
\mid & c! & \text{(send)} & \mid c? & \text{(receive)} \\
\mid & \mathsf{throw}\ e & \text{(throw)} & \mid \mathsf{fin}\ n & \text{(compensation)} \\
\mid & P;P & \text{(sequence)} & \mid P \sqcap P & \text{(choice)} \\
\mid & P \parallel P & \text{(parallel)} & \mid n[P,E,F] & \text{(scope)} \\
\mid & c_1?{\rightarrow}P_1 \,[\!]\, c_2?{\rightarrow}P_2 & \text{(guarded choice)} & &
\end{array}$$

$$E ::= \overline{e : P} \qquad\qquad F ::= P \qquad\qquad RP ::= [P,E]$$

The major difference from *Chor* is that it takes a local view on communications, where sending and receiving actions represent roles' local view of interactions. We would use the term "communication action" to denote either a sending or a receiving action. A sending action and a receiving action engage in a handshake when they have the same channel name and both roles involved are ready to perform them. Besides, here we use the normal non-deterministic choice, and introduce the guarded choice.

Another important difference from *Chor* is that we have scopes embedded in the processes, with its exception block E, and rename the "finalizer" to "compensation". These terms follow the WS-BPEL specification. Also, we have role process RP, which is used to represent independent roles.

The top structure in *Role* is the task S which is the parallel composition of a set of role processes on the set of local channels $\mathcal{CH}$.

$$S ::= \mathcal{CH} \bullet (\|_i\ [P_i, E_i])$$

4.2 Semantics

The trace semantics for *Role* language can be similarly defined as in Section 3. We introduce compensation context φ, which is a (possibly empty) sequence of compensation closures of the form $(n : F : \psi)$, where n is a scope name, F is the compensation block of n, and ψ is a compensation context that accumulates during performing process $n.1$.

We express the semantics of a process under some compensation context φ as a set of pairs with the form (s, φ'), where s represents a trace of the process, and

φ' represents the new compensation context after executing the process under φ. Initially, φ is empty.

The basic processes skip, a and $\mathsf{throw}\ e$ have no effect to the compensation context, so the semantics is trivial.

$$[\![\mathsf{skip}]\!]^{\varphi} \mathrel{\widehat=} \{(\langle\checkmark\rangle, \varphi)\} \qquad [\![a]\!]^{\varphi} \mathrel{\widehat=} \{(\langle a, \checkmark\rangle, \varphi)\} \qquad [\![\mathsf{throw}\ e]\!]^{\varphi} \mathrel{\widehat=} \{(\langle \Rsh_e\rangle, \varphi)\}$$

For the scope activity $n[P, E, F]$, if process P completes successfully, $\langle n : F : \psi\rangle$ will be inserted to the front of φ. Here ψ is the accumulated compensation closures during performing P. If P throws an exception, φ remains the same.

$$\begin{aligned}[\![n[P, E, F]]\!]^{\varphi} \ \mathrel{\widehat=}\ & \{(s\langle\checkmark\rangle, \langle n : F : \psi\rangle ^\frown \varphi) \mid (s\langle\checkmark\rangle, \psi) \in [\![P]\!]^{\langle\rangle}\} \ \cup \\ & \{(st, \varphi) \mid (s\langle\Rsh_e\rangle, \psi) \in [\![P]\!]^{\langle\rangle} \wedge (t, -) \in \mathit{hdl}(E, e)^{\psi}\}\end{aligned}$$

The function $\mathit{hdl}(E, e)^{\varphi}$ can be defined similarly as in Section 3.

$$\mathit{hdl}(E, e)^{\varphi} \mathrel{\widehat=} \begin{cases} [\![P]\!]^{\varphi} & \text{if } \mathsf{hd}(E) = e : P \vee \mathsf{hd}(E) = * : P \\ \mathit{hdl}(\mathsf{tl}(E), e)^{\varphi} & \text{if } \mathsf{hd}(E) = e' : P \wedge e' \neq e \\ \{(\langle\Rsh_e\rangle, -)\} & \text{if } E \text{ is empty} \end{cases}$$

The semantics of $\mathsf{fin}\ n$ is also similar:

$$[\![\mathsf{fin}\ n]\!]^{\varphi} \mathrel{\widehat=} \{(s\langle\checkmark\rangle, \varphi) \mid (s\langle\checkmark\rangle, -) \in \mathit{getf}(n, \varphi)\}$$

The semantics of choice is simple: $[\![P_1 \sqcap P_2]\!]^{\varphi} \mathrel{\widehat=} [\![P_1]\!]^{\varphi} \cup [\![P_2]\!]^{\varphi}$.

The semantic rules given above do not have much difference from what for *Chor*. Now we discuss the more interesting parts related to the communication and parallel structures. The technique used here is inspired by [5] to define the traces of parallel processes. Furthermore, the semantics for sequential composition is redefined, too.

In the forthcoming discussion, α ranges over the local actions and communications (e.g. $c!$ and $c?$). The trace terminal marks $\checkmark$ and $\Rsh_e$ are still used. Additionally, we introduce a new terminal mark δ_X to represent that the process gets stuck and waits to communicate along channels in X, where X is a power set of communication actions. In general, δ_X represents the interleaving of waiting to communicate. For instance, $\delta_{\{\{a?,b?\},\{c!\}\}}$ waits for either $a?$ or $b?$, or waits for $c!$ interleavingly. For simplification, we will write $\delta_{\{a?,b?\}}$ to represent $\delta_{\{\{a?,b?\}\}}$, and write $\delta_{a!}$ instead of $\delta_{\{\{a!\}\}}$. We use ϵ for the empty trace, and write st for the concatenation of t onto s, which is equal to s if s ends with δ_X.

For the sequential composition of P_1 and P_2, if P_1 ends with either $\Rsh_e$ or δ_X (raising exception or getting stuck), then P_2 does not execute.

$$\begin{aligned}[\![P_1; P_2]\!]^{\varphi} \ \mathrel{\widehat=}\ & \{(st, \chi) \mid (s\langle\checkmark\rangle, \psi) \in [\![P_1]\!]^{\varphi} \wedge (t, \chi) \in [\![P_2]\!]^{\psi}\} \ \cup \\ & \{(s\langle\tau\rangle, \psi) \mid (s\langle\tau\rangle, \psi) \in [\![P_1]\!]^{\varphi} \wedge \tau \in \{\Rsh_e, \delta_X\} \text{ for some } X\}\end{aligned}$$

A sending action $c!$ or receiving action $c?$ represents the potential for a process to perform communication. Action $c!$ may eventually succeed with trace $\langle c!, \checkmark\rangle$, which can be reduced to c with a parallel receiving action $c?$; or fail with trace $\langle \delta_{c!} \rangle$, which means that the sending will never succeed in the future (thus the process gets stuck). We have similar explanation to the receiving action.

$$[\![c!]\!]^{\varphi} \mathrel{\widehat{=}} \{(\langle c!, \checkmark \rangle, \varphi), (\langle \delta_{c!} \rangle, \varphi)\} \qquad [\![c?]\!]^{\varphi} \mathrel{\widehat{=}} \{(\langle c?, \checkmark \rangle, \varphi), (\langle \delta_{c?} \rangle, \varphi)\}$$

The semantics of guarded choice is defined as follows, where $\langle c_1? \rangle s$ denotes a trace composed by concatenation of action $c_1?$ and trace s.

$$\begin{aligned} [\![c_1? \rightarrow P \,[\!]\, c_2? \rightarrow Q]\!]^{\varphi} \;\; \widehat{=} \;\; & \{(\langle \delta_{\{c_1?, c_2?\}} \rangle, \varphi)\} \cup \{(\langle c_1? \rangle s, \varphi) \mid s \in [\![P]\!]^{\varphi}\} \cup \\ & \{(\langle c_2? \rangle s, \varphi) \mid s \in [\![Q]\!]^{\varphi}\} \end{aligned}$$

For the semantics of parallel composition of processes, we introduce some auxiliary definitions in the first. The predicate $\mathit{match}(\alpha_1, \alpha_2, c)$ indicates whether α_1 and α_2 are a pair of matching communication actions on channel c, i.e.

$$\mathit{match}(\alpha_1, \alpha_2, c) \mathrel{\widehat{=}} \begin{cases} \mathit{true} & \text{if } \{\alpha_1, \alpha_2\} = \{c?, c!\} \\ \mathit{false} & \text{otherwise} \end{cases}$$

For the parallel composition of traces, we distinguish two different cases: (1) at most one trace ends with δ_X; (2) both traces end with δ_X.

For the first case, we define:

$$s\langle \tau \rangle \parallel t\langle \tau' \rangle \;\widehat{=}\; \{r\langle \tau \oplus \tau' \rangle \mid r \in \mathit{merge}(s, t)\}$$

where τ and τ' are meta variables over terminal marks $\{\checkmark, \Rsh_e, \delta_X\}$. The terminal mark of parallel composition is shown in the table below.

$\oplus$	$\checkmark$	$\Rsh_{e_1}$	δ_X
$\checkmark$	$\checkmark$	$\Rsh_{e_1}$	δ_X
$\Rsh_{e_2}$	$\Rsh_{e_2}$	$\Rsh_{e_1 \uplus e_2}$	$\Rsh_{e_2}$

Function $\mathit{merge}(s, t)$ returns the set of all traces formed by merging s and t fairly, allowing synchronization of matching communications. We let $\mathit{merge}(s, \epsilon) = \mathit{merge}(\epsilon, s) = \{s\}$. When s and t are nonempty, their fair merge is defined inductively, where c in the trace denotes a handshake of $c!$ and $c?$.

$$\begin{aligned} \mathit{merge}(\langle \alpha_1 \rangle s_1, \langle \alpha_2 \rangle t_1) \;\widehat{=}\; & \{\langle \alpha_1 \rangle r \mid r \in \mathit{merge}(s_1, \langle \alpha_2 \rangle t_1)\} \cup \\ & \{\langle \alpha_2 \rangle r \mid r \in \mathit{merge}(\langle \alpha_1 \rangle s_1, t_1)\} \cup \\ & \{\langle c \rangle r \mid \mathit{match}(\alpha_1, \alpha_2, c) \wedge r \in \mathit{merge}(s_1, t_1)\} \end{aligned}$$

Thus we have $\langle c!, \checkmark \rangle \parallel \langle c?, \checkmark \rangle = \{\langle c, \checkmark \rangle, \langle c!, c?, \checkmark \rangle, \langle c?, c!, \checkmark \rangle\}$, and $\langle c!, \checkmark \rangle \parallel \langle \delta_{c?} \rangle = \{\langle c!, \delta_{c?} \rangle\}$.

For the second case, we define:

$$s\langle \delta_X \rangle \parallel t\langle \delta_Y \rangle \;\widehat{=}\; \begin{cases} \{\} & \text{if } \exists \alpha \in \bigcup X, \beta \in \bigcup Y, c \bullet \\ & \qquad \mathit{match}(\alpha, \beta, c) \\ \{r\langle \delta_{X \cup Y} \rangle \mid r \in \mathit{merge}(s, t)\} & \text{otherwise} \end{cases}$$

If there exists any matching stuck marks (e.g., $\delta_{\{\{a!\},\{b?,c?\}\}}$ and $\delta_{\{\{c!\},\{d?\}\}}$ are matched on channel c), then the set of traces of $s \parallel t$ is empty. This is because the merge should be fair: if one process has an action $c!$, another process has a $c?$, and neither of them communicate with other processes, then their parallel composition should not deadlock. In other words, a trace should never end with $\delta_{\{\{c!\},\{c?\}\}}$. We simply discard such "unfair" traces.

Otherwise, we wait for communication along channels in $X \cup Y$. Thus, we have $\langle \delta_{a!} \rangle \parallel \langle \delta_{a?} \rangle = \{\}$, and $\langle \delta_{a!} \rangle \parallel \langle \delta_{b?} \rangle = \{\langle \delta_{\{\{a!\},\{b?\}\}} \rangle\}$, which denotes the process waits to communication along the actions $a!$ and $b?$ forever.

The rule for parallel composition of processes is the same as in Section 2.2.

$$\begin{aligned} [\![P_1 \parallel P_2]\!]^{\varphi} \;\widehat{=}\; & \{(r, \chi) \mid r \in (s \parallel t) \;\wedge\; \chi \in (\mathit{interl}(\varphi', \varphi'') ^\frown \varphi) \;\wedge \\ & (s, \varphi' {}^\frown \varphi) \in [\![P_1]\!]^{\varphi} \;\wedge\; (t, \varphi'' {}^\frown \varphi) \in [\![P_2]\!]^{\varphi}\} \end{aligned}$$

As an example, we have $[\![c! \parallel c?]\!]^{\varphi} = \{(\langle c, \checkmark \rangle, \varphi), (\langle c?, c!, \checkmark \rangle, \varphi), (\langle c!, c?, \checkmark \rangle, \varphi),$ $(\langle c!, \delta_{c?} \rangle, \varphi), (\langle c?, \delta_{c!} \rangle, \varphi)\}$. The trace $\langle c, \checkmark \rangle$ denotes that the two actions communicate with each other. The trace $\langle c!, \delta_{c?} \rangle$ denotes that the sending action appearing on the left side of the parallel construct will eventually communicate with some other receiving action (but not the one on the right side), while the receiving action on the right side has to stuck because it cannot find a matching action. We define the semantics in this way so that compositionality is achieved – as an example, please simply consider the semantics of $c? \parallel c! \parallel c?$.

The semantics for a role process is similar to a root choreography:

$$\begin{aligned} [\![[P, E]]\!] \;\widehat{=}\; & \{s\langle \checkmark \rangle \mid (s\langle \checkmark \rangle, -) \in [\![P]\!]^{\langle\rangle}\} \;\cup \\ & \{st \mid (s\langle \Rsh_e \rangle, \varphi) \in [\![P]\!]^{\langle\rangle} \wedge (t, -) \in \mathit{hdl}(E, e)^{\varphi}\} \end{aligned}$$

It is easy to prove that the parallel composition and both forms of choice satisfies commutativity and associativity in the semantics above.

Finally we define the semantics of a task. We introduce $\mathit{close}_{\mathcal{CH}}(T)$ that "closes" all channels of $\mathcal{CH}$ in trace set T, in the sense that the channels in $\mathcal{CH}$ will not used for communication with outside. To achieve this, we take two steps: first, we exclude all the traces that include either $c!$ or $c?$, with them the result of the filter is empty. Then, we modify the stuck mark of the remaining traces by removing communications along channels in $\mathcal{CH}$.

$$\begin{aligned} \mathit{close}_{\mathcal{CH}}(T) \;&\widehat{=}\; \{\mathit{close}_1(t, \mathcal{CH}) \mid t \in T \wedge \forall c \in \mathcal{CH} \bullet t \downharpoonleft \{c!, c?\} = \langle\rangle\} \\ \mathit{close}_1(t, \mathcal{CH}) \;&\widehat{=}\; \begin{cases} t & \text{if } t = t'\langle \checkmark \rangle \vee t = t'\langle \Rsh_e \rangle \\ t'\langle \delta_{X|\mathcal{CH}} \rangle & \text{if } t = t'\langle \delta_X \rangle \end{cases} \end{aligned}$$

Here we define $X \,|\, \mathcal{CH} \;\widehat{=}\; \{A \mid \exists B \in X \bullet A = (B \setminus \mathcal{CH}) \wedge A \neq \emptyset\}$, where $B \setminus \mathcal{CH}$ removes all communications along channels in $\mathcal{CH}$ from B. For example, $\{\{c!\}\}|\{c\} = \{\}$, and $\{\{a?, b?\}, \{c!\}\}|\{b, c\} = \{\{a?\}\}$.

Thus, we have $\mathit{close}_{\{c\}}([\![[c!, \epsilon] \parallel [c?, \epsilon]]\!]) = \{\langle c, \checkmark \rangle\}$ and $\mathit{close}_{\{c\}}([\![[c!, \epsilon] \parallel$ $[\mathsf{skip}, \epsilon]]\!]) = \{\langle \delta_{\{\}} \rangle\}$, which denotes an internal deadlock.

The semantics of a task is simply defined as follows:

$$[\![\mathcal{CH} \bullet (\Vert_i \, [P_i, E_i])]\!] \;\widehat{=}\; \mathit{close}_{\mathcal{CH}}([\![\Vert_i \, [P_i, E_i]]\!])$$

Fig. 1. Endpoint Projection Rules

$\pi(\mathsf{skip}, i)$	$\widehat{=}$	skip	
$\pi(a^i, i)$	$\widehat{=}$	a	
$\pi(a^i, j)$	$\widehat{=}$	skip	when $j \neq i$
$\pi(c^{[i,j]}, i)$	$\widehat{=}$	$c^{[i,j]}!$	
$\pi(c^{[i,j]}, j)$	$\widehat{=}$	$c^{[i,j]}?$	
$\pi(c^{[i,j]}, k)$	$\widehat{=}$	skip	when $k \neq i \wedge k \neq j$
$\pi(\mathsf{throw}\ e, i)$	$\widehat{=}$	$\mathsf{throw}\ e$	
$\pi(\mathsf{perf}\ n, i)$	$\widehat{=}$	$n[\pi(n.1, i), \pi(n.2, i), \pi(n.3, i)]$	
$\pi(\mathsf{fin}\ n, i)$	$\widehat{=}$	$\mathsf{fin}\ n$	
$\pi(\overline{e : A}, i)$	$\widehat{=}$	$\overline{e : \pi(A, i)}$	
$\pi(A_1; A_2, i)$	$\widehat{=}$	$\pi(A_1, i); \pi(A_2, i)$	
$\pi(A_1 \parallel A_2, i)$	$\widehat{=}$	$\pi(A_1, i) \parallel \pi(A_2, i)$	
$\pi(A_1 \overset{i}{\sqcap} A_2, i)$	$\widehat{=}$	$\gamma_1; \pi(A_1, i) \sqcap \gamma_2; \pi(A_2, i)$	where $\begin{cases} \gamma_1 = \parallel_{j \in 1..n \wedge j \neq i} c'_j! \\ \gamma_2 = \parallel_{j \in 1..n \wedge j \neq i} c''_j! \end{cases}$
$\pi(A_1 \overset{i}{\sqcap} A_2, j)$	$\widehat{=}$	$c'_j? \rightarrow \pi(A_1, j) \mathbin{[\!]} c''_j? \rightarrow \pi(A_2, j)$	when $j \neq i$

Although the semantics seems complicated, we would point out that the complexity is rooted from the basic communication activities that any process algebra has, as discussed in Brookes's paper [5], rather than the exception handling and finalization constructs.

5 Projection

A projection is a procedure which takes a choreography specification in *Chor* and generates a set of processes in *Role*, where each process corresponds to a role in the choreography. No standard projection is defined in WS-CDL. In this section we give our projection rules, and discuss some issues related.

Firstly, we give a projection rule for the root choreography $[A, E]$, where A and E are projected to the process and exception block at each role process i.

$$\pi([A, E], i) \widehat{=} [\pi(A, i), \pi(E, i)]$$

The project rules for each form of activity is given in Fig. 1. The basic activity a^i generates action a at role R^i, or skip at other roles. The interactive activity $c^{[i,j]}$ generates sending action $c!$ and receiving action $c?$ at role R^i and R^j respectively. The rule for throw activity $\mathsf{throw}\ e$ is based on an assumption that each exception occurred in a choreography is global, which causes the same exception at every role. The activity $\mathsf{perf}\ n$ is projected to each role as a scope with name n, process $\pi(n.1, i)$, exception block $\pi(n.2, i)$, and compensation block $\pi(n.3, i)$, where $n.1$, $n.2$, and $n.3$ are the activity, exception block and finalizer of choreography n respectively. Note that this rule depends on the corresponding context Γ. Finalizing $\mathsf{fin}\ n$ generates the same action $\mathsf{fin}\ n$ at each role. Exception block $\overline{e : A}$ is simply projected to an exception block $\overline{e : \pi(A, i)}$ at role i. The rules for sequential and parallel compositions are trivial.

The most interesting rules are those for choice structure $A_1 \overset{i}{\sqcap} A_2$. For each role R^j ($j \neq i$), we should introduce two fresh channels, namely c'_j and c''_j. The projection of $A_1 \overset{i}{\sqcap} A_2$ on a role other than R^i takes the form of a guarded choice. On the other hand, the projection on role R^i is an ordinary choice with each branch beginning at a set of sending actions. As a result, when the execution of the roles arrives at their versions of the choice structure, role R^i makes the real choice, and notifies all the other roles on which branch it selects. Thus, all the roles will take the same branch in their versions of the choice consistently.

We illustrate a simple example of projection here.

Example 3. The choreography below involves two roles. After R^2 receives a message from R^1, it may either acknowledge R^1 and proceeds, or throw an exception so that the choreography is interrupted.

$$C = [c^{[1,2]}; (c^{[2,1]} \overset{2}{\sqcap} \mathsf{throw}\ e), \epsilon]$$

After projection, we get the following processes (we omit the scope here since the exception handler is empty):

$$P_1 = c^{[1,2]}!; (c'? \to c^{[2,1]}? \mathbin{[\!]} c''? \to \mathsf{throw}\ e) \quad P_2 = c^{[1,2]}?; ((c'!; c^{[2,1]}!) \sqcap (c''!; \mathsf{throw}\ e))$$

where c' and c'' are the fresh channels introduced in projection. It is not difficult to verify that $[\![\mathcal{CH} \bullet ([P_1, \epsilon] \parallel [P_2, \epsilon])]\!] \downharpoonleft acts(C) = [\![C]\!]$, where $\mathcal{CH} = \{c^{[1,2]}, c^{[2,1]}, c', c''\}$. $\square$

In the equation above, we use the filter operation $\downharpoonleft$ to restrict a trace (or a trace set) to mention only actions from a given action set. The notation $acts(C)$ denotes the set of all activities appearing in choreography C. This extra step removes the handshake actions of the fresh channels.

Example 4. The choreography C below illustrates concurrent exception:

$$C = [(a_1^1 \overset{1}{\sqcap} \mathsf{throw}\ e_1) \parallel (a_2^2 \overset{2}{\sqcap} \mathsf{throw}\ e_2), \epsilon]$$

After projection, we get the following processes:

$$\begin{aligned} P_1 &= ((c'_1!; a_1^1) \sqcap (c''_1!; \mathsf{throw}\ e_1)) \parallel (c'_2? \to \mathsf{skip} \mathbin{[\!]} c''_2? \to \mathsf{throw}\ e_2) \\ P_2 &= (c'_1? \to \mathsf{skip} \mathbin{[\!]} c''_1? \to \mathsf{throw}\ e_1) \parallel ((c'_2!; a_2^2) \sqcap (c''_2!; \mathsf{throw}\ e_2)) \end{aligned}$$

where c'_1, c''_1, c'_2 and c''_2 are four fresh channels.

Let $\mathcal{CH}$ include all the channel names, we can also verify that $[\![\mathcal{CH} \bullet ([P_1, \epsilon] \parallel [P_2, \epsilon])]\!] \downharpoonleft acts(C) = [\![C]\!]$, with the trace set:

$$\{\langle a_1^1, a_2^2, \checkmark\rangle, \langle a_2^2, a_1^1, \checkmark\rangle, \langle a_2^2, \Rsh_{e_1}\rangle, \langle a_1^1, \Rsh_{e_2}\rangle, \langle \Rsh_{e_1 \uplus e_2}\rangle\}$$

Please notice that $\langle \Rsh_{e_1}\rangle$ and $\langle \Rsh_{e_2}\rangle$ are not in the trace set, since we do not have forced termination.

We hope that the combination of processes can realize the behavior of the choreography. That is, for projection π and choreography C, we hope to prove the following equation:

$$[\![\mathcal{CH} \bullet (\pi(C,1) \parallel \cdots \parallel \pi(C,n))]\!] \downharpoonright acts(C) = [\![C]\!] \qquad (1)$$

where $\mathcal{CH}$ includes all the communication channels defined in the choreography and the fresh channels added by projection. In the previous examples, we already see this equation holds.

This equation says that if we "close" the set of traces generated by the parallel composition of all the role processes wrt the inter-role channels defined in C, and restrict the activities in each trace to the activity set of C, then the result should be equal to the set of traces of the choreography from which the role processes are projected. A formal proof of Equation (1) is an important future work.

6 Conclusion and Future Work

Web service choreography describes a global-view protocol for collaboration among multiple roles, while a set of suitable orchestrations can form a implementation of the protocol. Formal models of choreography and orchestration are important and useful in exploring the subtle features in languages such as WS-CDL and WS-BPEL, and the connection between them. In this paper, we continue the research initiated in [16], with special focus on exception handling and transactionality. Two languages *Chor* and *Role* for choreography and orchestration respectively are introduced, together with formal semantics. Corresponding projection rules are provided, too.

The main contributions of this paper are:

1. We present a denotational (trace) semantics for exception handling and finalization for the choreography language *Chor*. To the best of our knowledge, no work has been done in this area.
2. We also present a trace semantics for the role process language *Role*, where we introduce a "stuck" notation.
3. We provide a set of projection rules that form a map from the choreography language to the role process language. The projection is based on the similarity of *choreography* and *scope* constructs, and naturally projects a *choreography* to a *scope* at each role process. The concept *dominant role* is also vital in defining the projection.

The correctness of the projection should be investigated further, in the sense to ensures that the combination of the set of processes produced does realize the behavior described by the choreography. For this, we need to formally prove Equation (1). Additionally, we want also to extend the model to support variables, states, and contents of exchanged messages.

Acknowledgements. We would like to thank Shengchao Qin for many helpful comments.

References

1. SOAP service description language. http://ssdl.org.
2. Web services choreography description language version 1.0, 2005. http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/.
3. Business process execution language for web services, version 1.1, May 2003. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel.
4. M. Baldoni, C. Badoglio, A. Martelli, V. Patti, and C. Schifanella. Verifying the conformance of web services to global interaction protocols: a first step. In *Proc. of WS-FM'05, LNCS 3670*. Springer, 2005.
5. S. Brookes. Traces, pomsets, fairness and full abstraction for communicating processes. In *Proc. of CONCUR'02, LNCS 2421*. Springer, 2002.
6. R. Bruni, H. Melgratti, and U. Montanari. Theoretical foundations for compensations in flow composition languages. In *Proc. of POPL'05*. ACM Press, 2005.
7. N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. Choreography and orchestration: A synergic approach for system design. In *Proc. of ICSOC'05, LNCS 3826*. Springer, 2005.
8. N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. Choreography and orchestration conformance for system design. In *Proc. of Coordination'06, LNCS 4038*. Springer, 2006.
9. M. Butler, T. Hoare, and C. Ferreira. A trace semantics for long-running transactions. In *25 Years of CSP, LNCS 3525*. Springer, 2004.
10. M. Butler and S. Ripon. Executable semantics for compensating CSP. In *Proc. of WS-FM'05, LNCS 3670*. Springer, 2005.
11. M. Carbone, K. Honda, N. Yoshida, R. Milner, G. Brown, and S. Ross-Talbot. A theoretical basis of communication-centred concurrent programming, 2006. http://www.w3.org/2002/ws/chor/edcopies/theory/note.pdf.
12. X. Fu, T. Bultan, and J. Su. Conversation protocols: A formalism for specification and verification of reactive electronic services. In *Proc. of CIAA'03, LNCS 2759*. Springer, 2003.
13. J. Li, J. He, G. Pu, and H. Zhu. Towards the semantics for web services choreography description language. In *Proc. of ICFEM'06, LNCS 4260*. Springer, 2006.
14. G. Pu, H. Zhu, Z. Qiu, S. Wang, X. Zhao, and J. He. Theoretical foundations of scope-based compensation flow language for web service. In *Proc. of FMOODS'05, LNCS 4307*. Springer, 2006.
15. Z. Qiu, S. Wang, G. Pu, and X. Zhao. Semantics of BPEL4WS-like fault and compensation handling. In *Proc. of FM'05, LNCS 3582*. Springer, 2005.
16. Z. Qiu, X. Zhao, C. Chao, and H. Yang. Towards the theoretical foundation of choreography. Accepted by WWW'07. Available as a tech. report at http://www.is.pku.edu.cn/~fmows/.
17. X. Zhao, H. Yang, and Z. Qiu. Towards the formal model and verification of web services choreography description language. In *Proc. of WS-FM'06, LNCS 4184*. Springer, 2006.