Toward the Formalization of BPEL

Laila Boumlik and Mohamed Mejri(✉)

Department of Computer Science, Laval University, Quebec, Canada
laila.boumlik.1@ulaval.ca, Mohamed.Mejri@ift.ulaval.ca

Abstract. During the recent years, the composition, the orchestration and the interaction of web services have generated a great deal of interest for the Internet community. BPEL is the most important standard language for web services orchestration. It provides a variety of constructors allowing to build complex services from simple ones. However, many of its features are complex and source of a large misunderstanding due to the absence of formalization. For instance, understanding the Event Handler, the Fault Handler, the Compensation Handler and the Termination Handler (EFCT) of BPEL is a big challenge even for an experienced programmer.

This paper aims to formalize the EFCT fragment of BPEL using a process algebra called AV-BPEL specially defined to fit with the BPEL language and endowed with a small steps operational semantics.

Keywords: Web services · BPEL · Formal methods · Process algebra

1 Introduction

With the great success of Service Oriented Computing (SOC) [1], web services are becoming the main model for automated interaction between distributed systems on the Web. SOC is based on the composition of several services that are typically designed to interact with other web services through the Internet to form larger applications. Standard web service framework are essentially based on three technologies: SOAP [2], WSDL [3] and UDDI [4]. The WSDL (Web Service Description Language) provides an abstract language to describe messages to be exchanged between services. The SOAP (Simple Object Access Protocol) is a protocol for exchanging structured information between the involved parties and the UDDI (Universal Description Discovery and Integration) is used to publish and discover web services.

Having the possibility of constructing new web services by composing existing ones has opened a new interesting prospective and has significantly influenced the way industrial applications are developed. BPEL (Business Process Execution Language) [6] is the standard of web services composition and orchestration. Among the interesting features of this expressive language, we found the fault and the compensation handling mechanism. In fact, sometime it is useful to give the end-users the possibility to cancel some services if other fails. For example, if Alice wants to book a plane ticket and a hotel for a given period of time but

A. Norta et al. (Eds.): ICSOC 2015 Workshops, LNCS 9586, pp. 157–167, 2016.
DOI: 10.1007/978-3-662-50539-7_13

after booking the hotel she does not find an appropriate plane ticket, then it is desirable to allow her to cancel the hotel reservation. Compensation are introduced in BPEL to give the possibility to remedy a situation where an activity of a group fails whereas others terminated with success.

Even if the global ideas of the fault handler and the compensation handler are clear, it remains that there are a large number of particular cases that need to be handled: Which fault handler catches which error? Which compensation handler needs to be called after a given error? Which activities need to be stopped when an error occurs? What happens if an error occurs during a compensation? What happens if an error occurs in a fault handler itself? These are only a part of the particular situations that need to be clarified. Combined with other features, like event handler and termination handler, the number of these kind of questions explode justifying the urgent need of a suitable formalization for BPEL.

Contribution: The main contribution of this paper is a detailed formalization of the EFCT of BPEL, that address the above questions, based on a dedicated process algebra endowed with a small step operational semantics.

The remainder of this paper is organized as follows. Section 2 presents a simplified language that abstracts BPEL for XML syntax and formalizes it. Section 3 gives some related work. Finally, some concluding remarks are given in Sect. 4.

2 BPEL Formalisation

2.1 Syntax of AV-BPEL: Abstracted Version BPEL

Let b be a metavariable that ranges over Boolean expressions and Exp ranges over classical expressions like arithmetic and string expressions. The syntax of AV-BPEL, a fragment of $BPEL$ that is abstracted from XML syntax, is given by the BNF grammar given in Table 1.

A basic activity A contains an empty process (NOP action) 1, an exit process 0, a Boolean expression b, an assignment expression $x := Exp$, a throw action $\Rsh_e$, a send action $a!\overrightarrow{v}$ and a receive action $a?\overrightarrow{x}$.

A composed activity S can be any basic activity in A, a different composition of two activities or or a scope $<\mathcal{E}, F, C, T, E, S>_n$.

A process Pr (either $<\mathcal{E}, F, E, S>_n$ or $<\mathcal{E}, F, E, I>_n$) has a name n, a running environment $\mathcal{E}$, a fault handler F an event handler E and either a normal activity S or an instantiation bloc I.

An instantiation bloc I allows a service to fork an activity when a request of an operation is received ($a?\overrightarrow{x}$).

An event handler E always starts by waiting for some event (receiving a message or an alarm based on a "timing out").

A compensation handler (respectively a termination handler) can be a basic activity, an indication to compensate a particular scope $\$_n$ or $\$$, an indication to compensate all the reached scopes that terminated with success. It can also

Table 1. Syntax of BPEL.

A	$::=$	1	(Empty)
	$\mid$	0	(Exit)
	$\mid$	b	Condition
	$\mid$	$x := Exp$	(Assignement)
	$\mid$	$\Rsh_e$	(Throw)
	$\mid$	$a!\overrightarrow{v}$	(Send action)
	$\mid$	$a?\overrightarrow{x}$	(Receive action)
F	$::=$		Fault Handler
	$\mid$	A	Activity
	$\mid$	$\Lsh_e \mid \Lsh$	(catch e), (catch all)
	$\mid$	$\$_n \mid \$$	(compensate n), (compensate all)
	$\mid$	RS	(Root Scop)
	$\mid$	$F.F \mid F+F \mid F^*F \mid F\Vert_\gamma F$	
C, T	$::=$		Compensation and Termination Handler
	$\mid$	A	Activity
	$\mid$	$\$_n \mid \$$	(compensate n), (compensate all)
	$\mid$	RS	(Root Scop)
	$\mid$	$T.T \mid T+T \mid T^*T \mid T\Vert_\gamma T$	
E	$::=$		Event Handler
	$\mid$	$a?x.S$	OnEvent
	$\mid$	$\delta^n.S$	(OnAlarm)
	$\mid$	$E+E$	(Choice)
S	$::=$	A	
	$\mid$	$S_1.S_2$	(Sequential composition)
	$\mid$	S_1+S_2	(choice)
	$\mid$	$S_1\Vert_\gamma S_2$	(Merge, parallel composition)
	$\mid$	$S_1^*S_2$	(Iteration operator)
	$\mid$	$<\mathcal{E}, F, C, T, E, S>_n$	(Scope)
I	$::=$		(Insntantiation Bloc)
	$\mid$	$(a?\overrightarrow{x})^+.S$	(Receive with create Instance)
	$\mid$	$(\sum_{i\in\Theta} a_i?\overrightarrow{x_i}.S_i)^+$	(Pick Activity)
	$\mid$	I_1+I_2	(choice)
	$\mid$	$I_1\Vert_\gamma I_2$	(Merge, parallel composition)
	$\mid$	$<\mathcal{E}, F, C, T, E, I>_n$	(Scope)
RS	$::=$	$<\mathcal{E}, F, T, E, S>_n$	(Root Scop)
Pr	$::=$	$<\mathcal{E}, F, E, S>_n \mid <\mathcal{E}, F, E, I>_n$	(Process)
D	$::=$		(Deployment)
	$\mid$	P^m	(Processess)
	$\mid$	S^m	(Sequence)
	$\mid$	I^m	(Instantiation)
	$\mid$	$a\uparrow\overrightarrow{v} \Vert D$	(Message Transit)
	$\mid$	$D\Vert D$	(Process Parallel composition)

be a root scope RS (similar to a normal scope except that it cannot contains a compensation handler itself.).

A fault handler can be a basic activity, a catch for a named error ($\Lsh_e$), a catch all $\Lsh$, an indication to compensate a particular scope $\$_n$ or $\$$, an indication to compensate all the reached scopes that terminated with success. It can also be a root scope RS.

Besides the above constructors, we introduce a deployment D that does not belongs to the syntax of BPEL, it is useful to formalize asynchronous communications where the sent action is not blocking whereas the receiving is blocking.

2.2 Semantics

Let $\mathcal{D}$ be set of all possible deployments, and let P , Q and R range over $\mathcal{D}$. The operational semantics of AV-BPEL is defined by the transition relation $\longrightarrow \in \mathcal{D} \times \mathcal{A} \times \mathcal{D}$. Table 3 gives some standard rules that are true for all deployments P, P_1, Q and Q_1. Also, it uses the equivalent relation $\equiv \in \mathcal{D} \times \mathcal{D}$ defined by Table 2.

Table 2. Axiom of BPEL.

$P + Q \equiv Q + P$	(A_1)	$P \|_\gamma Q \equiv Q \|_\gamma P$	(A_2)
$(P + Q) + R \equiv P + (Q + R)$	(A_3)	$(P \|_\gamma Q) \|_\gamma R \equiv P \|_\gamma (Q \|_\gamma R)$	(A_4)
$0 + P \equiv P$	(A_5)	$1.P \equiv P$	(A_6)
$0 \|_\gamma P \equiv P$	(A_7)	$1 \|_\gamma P \equiv P$	(A_8)

We denote by $< \mathcal{E}, F, C^?, T^?, E, P >_n$ a root scope or a process, i.e., the termination handler and the compensation handler are optional.

•*Communication:* Sending values $\overrightarrow{v}$ on a channel a, denoted by $a!\overrightarrow{v}$, is not a blocking action and its produces a floating message, denoted by $a \upharpoonleft \overrightarrow{v}$, meaning that the message has not been received yet. Receiving messages on a channel a, denoted by $a?\overrightarrow{x}$ is however always blocking until seeing a floating message $a \upharpoonleft \overrightarrow{v}$ such that the correlation values in $\overrightarrow{x}$ match with the correlation values in $\overrightarrow{v}$. This matching is denoted by $\overrightarrow{v} \sqsubseteq \overrightarrow{x}$, where $(x_1, \ldots, x_n) \sqsubseteq (v_1, \ldots, v_n)$ if $x_i = v_i$ or x_i is a variable (i.e., there exists a substitution σ from variables to values such that $\sigma(\overrightarrow{x}) = \overrightarrow{v}$.).

To formalize this asynchronous communication, we introduce the relation $\longrightarrow\!\!\!\!\!\rightarrow \in \mathcal{D} \times \mathcal{A} \times \mathcal{D}$ that aims to show whether a process is ready to perform an action. This relation is not blocking even for the reception as shown in Table 4.

Now, the communication rules are formalized as shown hereafter.

$$(R^1_{Exp}) \frac{P \overset{x:=Exp}{\twoheadrightarrow} Q}{< \mathcal{E}, F, C^?, T^?, E, P >^m_n \xrightarrow{\tau} < \mathcal{E} \dagger [x \mapsto v], F, C^?, T^?, E, Q >^{m'}_n} [\![\mathcal{E}(Exp)]\!] = v$$

$$(R^1_b) \frac{P \overset{b}{\twoheadrightarrow} Q}{< \mathcal{E}, F, C^?, T^?, E, P >^m_n \xrightarrow{\tau} < \mathcal{E}, F, C^?, T^?, E, Q >^{m'}_n} [\![\mathcal{E}(b)]\!] = 1$$

Table 3. Classical Rules for $\longrightarrow$.

$$(R_{\equiv})\ \frac{P \equiv P_1 \quad P_1 \xrightarrow{a} P_2 \quad P_2 \equiv Q}{P \xrightarrow{a} Q}$$

$$(R_{\Rsh_e})\frac{\Box}{\Rsh_e \xrightarrow{\Rsh_e} 1} \qquad (R_{!})\frac{\Box}{a!\overrightarrow{v} \xrightarrow{a!\overrightarrow{v}} a\!\uparrow\overrightarrow{v}}$$

$$(R_{+})\frac{P \xrightarrow{a} P'}{P+Q \xrightarrow{a} P'} \qquad (R_{.})\frac{P \xrightarrow{a} P'}{P.Q \xrightarrow{a} P'.Q}$$

$$(R_{*}^{d})\frac{Q \xrightarrow{a} Q'}{P^{*}Q \xrightarrow{a} Q'} \qquad (R_{*})\frac{P \xrightarrow{a} P'}{P^{*}Q \xrightarrow{a} P'.(P^{*}Q)}$$

$$(R_{\|_\gamma})\frac{P \xrightarrow{a} P'}{P\|_\gamma Q \xrightarrow{a} P'\|_\gamma Q} a \notin \gamma \ (R_{\|_\gamma}^{S})\frac{P \xrightarrow{a!} P' \quad Q \xrightarrow{a?} Q'}{P\|_\gamma Q \xrightarrow{a} P'\|_\gamma Q'}\ a \in \gamma$$

Table 4. Definition of $\twoheadrightarrow$.

$$(R_{\equiv}^{\twoheadrightarrow})\ \frac{P \equiv P_1 \quad P_1 \overset{a}{\twoheadrightarrow} P_2 \quad P_2 \equiv Q}{P \overset{a}{\twoheadrightarrow} Q}$$

$$(R_{Exp})\frac{\Box}{x := Exp \overset{x:=Exp}{\twoheadrightarrow} 1} \quad (R_{b})\frac{\Box}{b \overset{b}{\twoheadrightarrow} 1} \quad (R_{?})\frac{\Box}{a?\overrightarrow{x} \overset{a?v}{\twoheadrightarrow} 1}$$

$$(R_{\uparrow})\frac{\Box}{a\!\uparrow\overrightarrow{v} \overset{a\uparrow\overrightarrow{v}}{\twoheadrightarrow} 1} \quad (R_{\Rsh_e}^{\twoheadrightarrow})\frac{\Box}{\Rsh_e \overset{\Rsh_e}{\twoheadrightarrow} 1} \quad (R_{!}^{\twoheadrightarrow})\frac{\Box}{a!\overrightarrow{v} \overset{a!\overrightarrow{v}}{\twoheadrightarrow} a\!\uparrow\overrightarrow{v}}$$

$$(R_{+}^{1})\frac{\Box}{(a?\overrightarrow{x})^{+}P \overset{(a?\overrightarrow{x})^{+}}{\twoheadrightarrow} P} \quad (R_{+}^{2})\frac{\Box}{(\sum_{i\in\Theta} a_i?\overrightarrow{x_i}.S_i)^{+} \overset{(a_i?\overrightarrow{x_i})^{+}}{\twoheadrightarrow} S_i}$$

$$(R_{+}^{\twoheadrightarrow})\frac{P \overset{a}{\twoheadrightarrow} P'}{P+Q \xrightarrow{a} P'} \qquad (R_{.}^{\twoheadrightarrow})\frac{P \overset{a}{\twoheadrightarrow} P'}{P.Q \overset{a}{\twoheadrightarrow} P'.Q}$$

$$(R_{*}^{d\twoheadrightarrow})\frac{Q \overset{a}{\twoheadrightarrow} Q'}{P^{*}Q \overset{a}{\twoheadrightarrow} Q'} \qquad (R_{*}^{\twoheadrightarrow})\frac{P \overset{a}{\twoheadrightarrow} P'}{P^{*}Q \overset{a}{\twoheadrightarrow} P'.(P^{*}Q)}$$

$$(R_{\|_\gamma}^{\twoheadrightarrow})\frac{P \overset{a}{\twoheadrightarrow} P'}{P\|_\gamma Q \overset{a}{\twoheadrightarrow} P'\|_\gamma Q} a \notin \gamma \ (R_{\|_\gamma}^{S\twoheadrightarrow})\frac{P \overset{a!}{\twoheadrightarrow} P' \quad Q \overset{a?}{\twoheadrightarrow} Q'}{P\|_\gamma Q \overset{a}{\twoheadrightarrow} P'\|_\gamma Q'}\ a \in \gamma$$

$$(R^{!}_{\uparrow})\frac{P \overset{a\uparrow\overrightarrow{v}}{\twoheadrightarrow} Q}{<\mathcal{E},F,C^?,T^?,E,P>^m_n \xrightarrow{\mathcal{E}(a)\uparrow\overrightarrow{\mathcal{E}(v)}} <\mathcal{E}',F',C'^?,T'^?,E',Q>^{m'}_n \parallel \mathcal{E}(a)\uparrow\overrightarrow{\mathcal{E}(v)}}$$

$$(R^{?}_{\uparrow})\frac{P \overset{a?\overrightarrow{x}}{\twoheadrightarrow} Q \quad D \overset{a\uparrow\overrightarrow{v}}{\twoheadrightarrow} D'}{<\mathcal{E},F,C^?,T^?,E,P>^m_n \parallel D \xrightarrow{a?\overrightarrow{v}} <\mathcal{E}\dagger(\overrightarrow{x}\mapsto\overrightarrow{v}),F,C^?,T^?,E,Q>^{m'}_n \parallel D'}\overrightarrow{v}\sqsubseteq\overrightarrow{x}$$

where $m, m' \in \{F, C, FT0, FT1, TC, \}$, and $[\![-]\!]$ is a function that evaluates an expression (Boolean expression, arithmetic expression, string expression, etc.). For the moment, we are not interested by having more details about it. The operator † is defined as follows:

$$\mathcal{E}\dagger(x\mapsto v)(y) = \begin{cases}\mathcal{E}(y) & \text{if } y \neq x\\ v & \text{if } y = x\end{cases}$$

•*Creating an Instance of a Process:* The sysntax of BPEL gives two possibilities to explicitly specify the fact that we want to create an instance of a process when a message is received: first with the parameter `createInstance=YES` used within the definition of a "Receive" action. This case is transformed to $(a?\overrightarrow{x})^+$ in our syntax. Or by using a "Pickup" activity that contains only "OnMessage" (same as Receive, but it does not have the `createInstance` option). This case is specified by $(\sum_{i\in\Theta} a_i?x_i.S_i)^+$ in our abstracted syntax.

The operational rule showing the creation of an instance are as shown here after.

$$(R^{4}_{+})\frac{I \overset{(a?\overrightarrow{x})^+}{\twoheadrightarrow} I' \quad D \overset{a\uparrow\overrightarrow{v}}{\twoheadrightarrow} D'}{<\mathcal{E},F,E,I>^m_n \parallel D \xrightarrow{a?\overrightarrow{v}} <\mathcal{E},F,E,I>^m_n \parallel <\mathcal{E}'\dagger(\overrightarrow{x}\mapsto\overrightarrow{v}),F',E',I'>^{m'}_n \parallel D'}\overrightarrow{v}\sqsubseteq\overrightarrow{x}$$

•*Principal Activity:* Some basic rules showing the evolution of an activity are shown by Table 3. Further behaviors are dictated by specific rules described hereafter.

– Each error thrown by the principal activity should be caught by the fault handler of the same scope.

$$(R^{2}_{A})\frac{P \xrightarrow{\Rsh_e} Q \quad F \xrightarrow{\Lsh_e} F'}{<\mathcal{E},F,C^?,T^?,E,P>^{FT}_n \xrightarrow{\tau} <\mathcal{E},F',C^?,T^?,E,Q>^{FT0}_n}$$

– If a throw $\Rsh_e$ doesn't have a dedicated catch in the fault handler, then it should be handled by the "catch all" activity.

$$(R^{3}_{A})\frac{P \xrightarrow{\Rsh_e} Q \quad F \overset{\Lsh_e}{\not\longrightarrow} \quad F \xrightarrow{\Lsh} F'}{<\mathcal{E},F,C^?,T^?,E,P>_n \xrightarrow{\tau} <\mathcal{E},F',C^?,T^?,E,Q>^{FT0}_n}$$

– If there is neither a specific catch nor a "catch all" activity for a specific error, the fault handler execute the "Compensate all" activity ($).

$$(R^{4}_{A})\frac{P \xrightarrow{\Rsh_e} Q \quad F \overset{\Lsh_e}{\not\longrightarrow} \quad F \overset{\Lsh}{\not\longrightarrow}}{<\mathcal{E},F,C^?,T^?,E,P>_n \xrightarrow{\tau} <\mathcal{E},\$,C^?,T^?,E,Q>)n^{FT0}}$$

– When the principal activity of a scope terminates with success, the scope is stored, in a compensation mode, for an eventual future compensation.

$$(R_A^5)\frac{< \mathcal{E}, F, C, T^?, E, P >_n \xrightarrow{a} < \mathcal{E}', F', C, T'^?, E', 1 >_n}{< \mathcal{E}, F, C, T^?, E, P >_n .S \xrightarrow{a} S. < \mathcal{E}', F', 1, T'^?, E', C >_n^C}$$

$$(R_A^6)\frac{< \mathcal{E}, F, C, T^?, E, P >_n \xrightarrow{a} < \mathcal{E}', F', C, T'^?, E', 1 >_n}{< \mathcal{E}, F, C, T^?, E, P >_n \xrightarrow{a} < \mathcal{E}', F', 1, T'^?, E', C >_n^C}$$

– In a compensation mode, a scope is executed like in a normal mode, except that we do not need to store it anymore when it terminates with success. More precisely:

- If the activity didn't finished yet, the compensation mode behaves like normal mode.

$$(R_C^1)\frac{< \mathcal{E}, F, C, T^?, E, P >_n \xrightarrow{a} < \mathcal{E}', F', C', T'^?, E', Q >_n}{< \mathcal{E}, F, C, T^?, E, P >_n^C \xrightarrow{a} < \mathcal{E}', F', C', T'^?, E', Q >_n^C} Q \neq 1$$

- If the activity of the compensation mode terminates, the scope is uninstalled.

$$(R_C^2)\frac{< \mathcal{E}, F, C, T^?, E, P >_n \xrightarrow{a} < \mathcal{E}', F', C', T'^?, E', 1 >}{< \mathcal{E}, F, C, T^?, E, P >_n^C \xrightarrow{a} 1}$$

•*Fault Handler, Termination Handler and Compensation Handler:* When an error occurs during the execution of the main activity of a scope, the control is passed to the fault handler which executes a list of treatments in the following order:

1. It stops all the running activities of the scope by the help of the relation $\longrightarrow_t$. The termination produces always a silent action τ.

$$(R_{FT}^1)\frac{P \xrightarrow{\tau}_t Q}{< \mathcal{E}, F, C^?, T^?, E, P >_n^{FT0} \quad \xrightarrow{\tau} \quad < \mathcal{E}, F, C^?, T^?, E, Q >_n^{FT0}}$$

– Stopping a scope consists on terminating its principal activity.

$$(R_T^1)\frac{P \xrightarrow{\tau}_t Q}{< \mathcal{E}, F, C^?, T^?, E, P >_n \quad \xrightarrow{\tau}_t \quad < \mathcal{E}, F, C^?, T^?, E, Q >_n}$$

– Terminating two parallel activities turns to terminate each one of them.

$$(R_T^2)\frac{P_1 \xrightarrow{\tau}_t Q_1}{P_1 \| P_2 \quad \xrightarrow{\tau}_t \quad Q_1 \| P_2}$$

– Terminating an iterative activity consists in removing its code.

$$(R_T^3)\frac{\square}{P_1^* P_2 \quad \xrightarrow{\tau}_t \quad 1}$$

- Stopping a sequence turns to remove all it code except those related to a compensation.

$$(R_T^4)\frac{}{P_1.P_2 \quad \xrightarrow{\tau}_t \quad P_2} P_1 \neq < \mathcal{E}, F, C^?, T^?, E, 1 >$$

2. Once an activity is stopped, the termination handler of the scope is executed.

$$(R_T^6)\frac{P \not\longrightarrow_t}{< \mathcal{E}, F, C, T^?, E, P >_n^{FT0} \quad \xrightarrow{\mathcal{E}(a)} \quad < \mathcal{E} \dagger a, F', C, T^?, E, P >_n^{FT1}}$$

- The termination handler can compensate activities

$$(R_{TC}^1)\frac{T \xrightarrow{\$} T'}{< \mathcal{E}, F, C^?, T, E, S >_n^{FT1} \quad \xrightarrow{\$} \quad < \mathcal{E}, F, C, T', E, S^{-1} >_n^{TC}}$$

- The activity of a compensation handler is executed via the relation $\longrightarrow_c$ and it can be invoked only by a fault handler, a termination handler or another compensation handler.

$$(R_{TC}^2)\frac{< \mathcal{E}, F, C^?, T, E, P >_n^{C} \quad \xrightarrow{a}_c \quad < \mathcal{E}', F', C'^?, T', E', Q >_n^{C}}{< \mathcal{E}, F, C^?, T, E, P >_n^{TC} \quad \xrightarrow{a} \quad < \mathcal{E}', F', C'^?, T', E', Q >_n^{TC}} a \neq \Rsh_n \text{ and } Q \neq 1$$

- Once the compensation is successfully terminated, the control is turned back to the calling activity (fault handler).

$$(R_{TC}^3)\frac{< \mathcal{E}, F, C^?, T, E, P >_n^{C} \quad \xrightarrow{a}_c 1}{< \mathcal{E}, F, C^?, T, E, P >_n^{TC} \quad \xrightarrow{a} \quad < \mathcal{E}, F, C^?, T, E, 1 >_n^{TF1}} a \neq \Rsh_e$$

- If an error occurs during the execution of a compensation handler, its remaining activity is ignored and the control is turned back to the calling fault handler without reporting it.

$$(R_{TC}^4)\frac{< \mathcal{E}, F, C^?, T, E, P >_n^{C} \xrightarrow{\Rsh_n}_c S'}{< \mathcal{E}, F, C^?, T, E, P >_n^{TC} \quad \xrightarrow{\tau} \quad < \mathcal{E}, F, C^?, 1, E, 1 >_n^{F}}$$

- The termination handler can run activities others than calling a compensation handler, but it can never signal an error.

$$(R_{TC}^5)\frac{< \mathcal{E}, F, C^?, T^?, E, T >_n \quad \xrightarrow{a} \quad < \mathcal{E}', F, C^?, T^?, E, T' >_n}{< \mathcal{E}, F, C^?, T, E, P >_n^{FT1} \quad \xrightarrow{a} \quad < \mathcal{E}', F, C, T', E, P >_n^{FT1}} a \neq \Rsh_e$$

3. Once the termination handler finished, the fault handler runs it own activity.

$$(R_F^5)\frac{T \not\twoheadrightarrow}{< \mathcal{E}, F, C^?, T, E, P >_n^{FT1} \quad \xrightarrow{\tau} \quad < \mathcal{E}, F, C^?, T, E, P >_n^{F}}$$

- The fault handler can compensate an activity if this is specified in its activity.

$$(R_{FC}^1)\frac{F \xrightarrow{\$} F'}{< \mathcal{E}, F, C^?, T^?, E, P >_n^{F} \quad \xrightarrow{\$} \quad < \mathcal{E}, F', C^?, T^?, E, P^{-1} >_n^{FC}}$$

where

$$P^{-1} = \begin{cases} Q^{-1}.a & \text{if } P = a.Q \\ Q^{-1} + R^{-1} & \text{if } P = Q + R \\ Q^{-1} \parallel R^{-1} & \text{if } P = Q \parallel R \\ P & \text{else} \end{cases}$$

- The code of the compensation handler is executed by the relation $\longrightarrow_c$.

$$(R^2_{FC})\frac{<\mathcal{E}, F, C^?, T^?, E, P>^C_n \quad \xrightarrow{a}_c \quad <\mathcal{E}', F', C'^?, T'^?, E', Q>^C_n}{<\mathcal{E}, F, C^?, T^?, E, P>^{FC}_n \quad \xrightarrow{a} \quad <\mathcal{E}', F', C'^?, T'^?, E', Q>^{FC}_n} a \neq \Rsh_e \text{ and } Q \neq 1$$

- Once a compensation handler terminate with success, the fault handler continues its activity.

$$(R^3_{FC})\frac{<\mathcal{E}, F, C^?, T^?, E, P>^C_n \quad \xrightarrow{a} \quad 1}{<\mathcal{E}, F, C^?, T^?, E, P>^{FC}_n \quad \xrightarrow{\mathcal{E}(a)} \quad <\mathcal{E} \dagger a, F, C^?, T^?, E, 1>^F_n} a \neq \Rsh_e$$

– A fault handler can run activities different from compensations.

$$(R^1_F)\frac{<\mathcal{E}, F, C^?, T^?, E, F>_n \quad \xrightarrow{a} \quad <\mathcal{E}', F, C^?, T^?, E, F'>_n}{<\mathcal{E}, F, C^?, T^?, E, P>^F_n \quad \xrightarrow{a} \quad <\mathcal{E}', F', C^?, T^?, E, P>^F_n} a \neq \Rsh_e$$

– A process or a scope that produces an error fails (i.e. becomes 0) regardless of its fault handler terminates with success or not as shown here after.

- If an error occurs inside a compensation handler, the fault handler propagates it to the parent scope and the current scope fails.

$$(R^4_{FC})\frac{<\mathcal{E}, F, C^?, T^?, E, P>^C_n \quad \xrightarrow{\Rsh_e}_c Q}{<\mathcal{E}, F, C, T^?, E, P>^{FC}_n \quad \xrightarrow{\Rsh_e} \quad 0}$$

- If an error occurs inside a fault handler itself, the error is propagated to the parent scope.

$$(R^2_F)\frac{F \xrightarrow{\Rsh_e} F'}{<\mathcal{E}, F, C^?, T^?, E, P>^F_n \quad \xrightarrow{\Rsh_e} \quad 0}$$

- If a fault handler terminates with success, its scope or process terminates in a fail mode (0).

$$(R^3_F)\frac{F \xrightarrow{a} 1}{<\mathcal{E}, F, C^?, T^?, E, P>^F_n \quad \xrightarrow{\mathcal{E}(a)}_c \quad 0} a \neq \Rsh_e$$

•*Event Handler:*

– Once an event is received, the activity of it corresponding event handler is executed in parallel with the principal activity of the scope.

$$(R_E)\frac{E \quad \overset{a?\overrightarrow{x}}{\twoheadrightarrow} \quad E' \quad D \overset{a\uparrow\overrightarrow{v}}{\twoheadrightarrow} D'}{<\mathcal{E}, F, C^?, T^?, E, P>^m_n \parallel D \quad \xrightarrow{a?\overrightarrow{v}} \quad <\mathcal{E} \dagger (\overrightarrow{x} \mapsto \overrightarrow{v}), F, C^?, T^?, E, E' \| P>^{m'}_n \parallel D'} \overrightarrow{v} \sqsubseteq \overrightarrow{x}$$

– Each error produced by an event handler is treated by the fault handler of the corresponding scope.

$$(R^{\Rsh}_E)\frac{E \xrightarrow{\Rsh_n} E' \quad F \xrightarrow{\Lsh_n} F'}{<\mathcal{E}, F, C^?, T^?, E, P> \xrightarrow{\tau} <\mathcal{E}, F', C^?, T^?, E, P>^{FT0}}$$

3 Related Work

Several interesting contributions were appeared during the last years to formalize the semantics of BPEL. Most of them tried to match the constructors of BPEL with some terms in an existing process algebra and only few works have attempted to define a new algebra that fit better with BPEL's features. In [7–9] the authors have presented a two-way mapping between a fragment of BPEL and the process algebra LOTOS. They have considered most of BPEL's activities including fault, compensation and event handlers. In [5] the authors have proposed a mapping from a BPEL process to a π-based calculus, named web$\pi\infty$. They advocated that the different mechanisms (fault handler, termination handler and compensation handler) for error handling are not needed and they proposed the idea of event notification to substitute them. The authors of [11] have proposed an approach based on two-way mapping between the π-based orchestration calculus and BPEL. [10] presented a formalization of BPEL 2.0 based on the π-calculus.The authors of [12] proposed an approach based RECATNet to model and verify a fragment of BPEL processes. The authors of [13] present a novel correct-by-construction formal approach based on refinement using the Event-B method. A Petri net sematics for BPEL has also been introduced in [14]. The main drawback of these works is that they are either too abstract to handle most of the interesting details related to BPEL or they lead to complex representation of some intuitive aspects like compensation. In [8,15,16], Pu et al. introduced an interesting process algebra to specify the activities of BPEL. Mostly, the authors focus on fault and compensation handling. Several other interesting formal semantics were proposed in the literature including [17,18] to formalize handlers in BPEL, but no one of them take into account all the details. Some of other advantages of AV-BPEL semantics are:

– Compensations can be executed in parallel in some cases as specified by the semantic of BPEL. But, this is not the case for [17,18].
– Receive action is blocking while send action is not which reflect better reality. Floating message give a simple way to integrate this kind of communication.

4 Conclusion

This paper introduces a process algebra called AV-BPEL to disambiguate some important but very confusing features of BPEL. These include fault handler, compensation handler, termination handler and event handler (EFCT). Despite that the global idea of these handlers can be easily understood, their full semantics hide many details making them the major source of difficulty behind BPEL. Compared to existing related work, the main advantage of our formalization is that it shows the real complexity hidden behind EFCT aspects by taking into account almost all details related to them.

References

1. Huhns, M.N., Singh, M.P.: Service-oriented computing: key concepts and principles. In: IEEE Internet Computing, pp. 75–81 (2005)
2. SOAP Version 1.2 Part 0: Primer, 2nd edn. (2007)
3. Web Services Description Language (WSDL 1.1) (2003)
4. UDDI Version 2.04 API Specification (2002)
5. Lucchi, R., Mazzara, M.: A pi-calculus based semantics for WS-BPEL. J. Logic Algebraic Program. **70**, 96–118 (2007)
6. Web Services Business Process Execution Language Version 2.0 (2007)
7. Salaün, G., Ferrara, A., Chirichiello, A.: Negotiation among web services using LOTOS/CADP. In: (LJ) Zhang, L.-J., Jeckle, M. (eds.) ECOWS 2004. LNCS, vol. 3250, pp. 198–212. Springer, Heidelberg (2004)
8. Mateescu, R., Poizat, P., Salaun, G.: Adaptation of service protocols using process algebra and on-the-fly reduction techniques. In: IEEE Transactions on Software Engineering. Institute of Electrical and Electronics Engineers (IEEE) (2012)
9. Ferrara, A.: Web services: a process algebra approach. In: Proceedings of 2nd ACM International Conference on Service Oriented Computing (2004)
10. Abouzaid, F., Mullins, J.: A calculus for generation, verification and refinement of BPEL specifications. In: Electronic Notes in Theoretical Computer Science, vol. 4421, pp. 43–65 (2008)
11. Yang, C., Zhong, F.: Towards the formal foundation of orchestration process. In: 5th ICCCNT- 2014 (2014)
12. Kheldoun, A., Ioualalen, M.: Transformation BPEL processes to recatnet for analysing web services compositions. In: Model-Driven Engineering and Software Development (MODELSWARD), pp. 425–430 (2014)
13. Guillaume BABIN, Y.A.A., PANTEL, M.: Formal verification of runtime compensation of web service compositions: a refinement and proof based proposal with Event-B. In: 2015 IEEE International Conference on Services Computing, vol. 978, pp. 7–15 (2015)
14. Stahl, C.: A Petri Net Semantics for BPEL, Humboldt-Universitat zu Berlin, Informatik-Berichte 188 (2005)
15. Pu, G., Zhu, H., Qiu, Z., Wang, S.-L., Zhao, X., Kleinberg, R.D.: Theoretical foundations of scope-based compensable flow language for web service. In: Gorrieri, R., Wehrheim, H. (eds.) FMOODS 2006. LNCS, vol. 4037, pp. 251–266. Springer, Heidelberg (2006)
16. Chirichiello, A., Salaun, G.: Encoding process algebraic descriptions of web services into BPEL. Web Intell. Agent Syst. Int. J. **5**(3), 419–434 (2007)
17. Lapadula, A., Pugliese, R., Tiezzi, F.: A calculus for orchestration of web services. In: Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 33–47. Springer, Heidelberg (2007)
18. Spieler, D.: Scope-based fct-handling in WS-BPEL 2.0. Master's thesis, Saarland University (2008)