Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 180 (2007) 55–70
www.elsevier.com/locate/entcs

A Formal Framework for Web Services Coordination

Claudio Guidi, Roberto Lucchi and Manuel Mazzara

Department of Computer Science, University of Bologna, Via Mura Anteo Zamboni 7 - 40127 Bologna, Italy
E-mail: {cguidi, lucchi, mazzara}@cs.unibo.it

Abstract

Recently the term Web Services Choreography has been introduced to address some issues related to Web Services Composition and Coordination. Several proposals for describing Choreography for Business Processes have been presented in the last years and many of these languages (e.g. BPEL4WS) make use of concepts as long-running transactions and compensations for coping with error handling. However, the complexity of BPEL4WS makes it difficult to formally define this framework, thus limiting the formal reasoning about the designed applications. In this paper, we formally address Web Services Coordination with particular attention to Web transactions. We enhance our past work - the Event Calculus - introducing two main novelties: i) a multicast event notification mechanism, and ii) event scope names binding. The former enables an easier specification of complex coordination scenarios — such as E-commerce applications require — while the latter allows many new interesting behaviors which can be very useful in business scenarios: the introduction of private event scope names — used to deal with security and privacy — and a dynamic event scopes definition that can be used to manage multiple instances of the same application.

Keywords: Web Services, coordination, long-running transactions, π-calculus, event notification.

1 Introduction

Web Services technology is a platform on which we can develop applications taking advantage of the Internet infrastructure. A Web Service, specifically, describes particular business functionalities that a company wants to expose through the Internet with the purpose of providing to other companies a way for using them. The key is on-the-fly software creation through the use of loosely coupled, reusable software components. Web Services promises to facilitate automated application-level business integration using the ease of connectivity to and global presence of the Internet infrastructure and replacing proprietary interfaces and data formats with a standard web-messaging infrastructure exploiting XML [26] technology. Although Web messaging is sufficient for some simple application integration needs, it does not adequately support the complete automation of critical business processes. The first generation of Web Services technology has largely focused on the web-messaging foundation supported by SOAP [20] and WSDL (Web Services

doi:10.1016/j.entcs.2006.10.046

Definition Language) [6]. WSDL is used to describe a Web Service in terms of its ports (addresses implementing this service), port types (the abstract definition of operations and exchanges of messages), and bindings (the concrete definition of packaging and transportation protocols, such as SOAP, are used to inter-connect two conversing end points). Although this foundation has the ability to specify critical information and requirements relating to the business process context, it does not support business processes that cross organizational boundaries. To truly integrate business processes across enterprise boundaries, merely supporting simple interaction using standard messages and protocols is insufficient. Business interactions require long-running interactions driven by an explicit process model. This raises the need for Web services composition languages also known as Web Services flow languages or Web Services orchestration/choreography languages.

Recently the terms Web Services choreography have been introduced to identify Web Services Composition and Coordination, that is the way of defining a complex service out of simpler ones. Several proposals for describing Choreography for business process have been presented in the last years: for example BPML [2], IBM's WSFL [11], Microsoft's XLANG ([21], [15]) or the more recent BPEL4WS [1] (which represents a trade-off between IBM and Microsoft). A Business Process Choreography consists of the aggregation of Web Services by encoding business rules or patterns governing services interactions and has the ability to reuse the created aggregations [8]. Business logic can be seen as the ingredient that sequences, coordinates, and manages interactions among Web Services. To program a complex cross-enterprise workflow task or business transactions, for example, it is possible to logically chain discrete Web Service activities into inter-enterprise business processes. In the world of Web Services a business process specifies the potential execution order of operations originating from a collection of Web Services, the shared data passed between these services, the trading partners that are involved in the joint process, their roles with respect to the process, joint exception handling conditions for the collection of Web Services and other factors that may influence how Web Services or organizations participate in a process [12]. This allows, in particular, specifying transactions between Web Services in order to increase the consistency and reliability of business processes that are composed out of Web Services.

Business process choreography requirements involve asynchronous interactions, flow coordination, business transaction activity and management. These are common to all business applications that need to coordinate multiple Web Services into a multi-step business transaction. Thus, the Web Services environment requires that several Web Service operations have transactional properties and be treated as a single logical unit of work performed as part of a business transaction. A business transaction is a consistent change in the state of the business that is driven by a well-defined business function. Usually, a business process is composed of several business transactions. In a Web Service environment business transactions essentially signify transactional Web Service interactions between organizations in order to accomplish some well-defined shared business objective. For example, consider,

a manufacturer that develops Web Service based solutions to automate the order and delivery business functions with its suppliers as part of a business transaction. The transaction between the manufacturer and its suppliers may only be considered as successful once all parts are delivered to their final destination, which could be days or weeks after the placement of the order.

In a web services environment transactions are complex, involve multiple parties, span many organizations, and can have long duration. More specifically, business transactions are automated long-running propositions involving negotiations, commitments, contracts, shipping and logistics, tracking, varied payment instruments, and exception handling. Performance of these business related tasks requires the infusion of transactional properties onto the Web Services paradigm. Although extremely reliable, the use of classic ACID transactions makes sense only when trusted parties are involved over short periods of time. Strict ACIDity is not appropriate to a loosely coupled world of autonomous trading partners, where security and inventory control issues prevent hard locking of local resources. Business applications require transactional support beyond classical ACID transactions.

We refer to nonACID transactions as Long Running Transactions. Error Handling in this context relies on the concept of Compensation. Most of the existing Choreography languages use long running transactions and compensations as a mechanism for describing loosely-coupled activities. Compensations are application-specific activities which attempt to reverse the effects of a previous activity carried out as part of a larger unit of work which is being abandoned. While for ACID transactions in databases the transaction coordinator and the resource it controls know all the uncommitted updates and have the full control on the order in which they must be reversed, in the case of business transactions the compensation behavior is itself a part of the business logic and must be explicitly specified.

It is straightforward to understand that, meeting these requirements, Choreography languages like XLANG or BPEL are quite complex. This complexity makes it difficult to understand the precise semantics, thus limiting the formal reasoning about the designed applications. With this motivation our work here is focused on formally addressing Web Services Coordination, with particular attention to Web transactions. In this paper we enhance our past work - the Event Calculus - which was based on the idea of event notification as the only error handling mechanism in Web Services Choreography. While the previous work were addressed on the problem of defining a sort of kernel language for modelling Web Service orchestration and error handling mechanisms, here we have extended the language introducing two main novelties: i) a multicast event notification mechanism, and ii) an event scope names binding mechanism. The first extension allows an easier specification of complex coordination scenarios (such as e-commerce applications) with respect to the algebra we presented in our past work which was focused mainly on error handling mechanisms unification. The second extension also allows many new interesting behaviours which can be very useful in business scenarios, such as the opportunity of handling security and privacy issues or the dynamic event scope definition for managing multiple instances of the same application. This feature shares some

similarities with the technique proposed by BPEL for the same purpose.

In this paper, we will proceed in the following way. In section 2 the formal approach to Web Services Coordination will be introduced sketching the state of the art in Web Services Choreography and comparing different programming models and the relative features. We decide to chose the π-calculus and we propose an extension in order to include transactional facilities. In section 3 we present this extension, the Event Calculus, with its syntax and semantics. Then, in section 4, we will propose an E-Commerce transactional scenario and we will formalize it using the Event Calculus in order to understand the potentialities of the language. Finally, in Section 5 we describe some related work reporting some conclusive remarks and possible future works.

2 A Formal Approach to Web Services Coordination

The problem of choreographing web services is tackled by a trio of standards that have been recently proposed to handle this next step in the evolution of Web services technology. The standards that support business process orchestration are: Business Process Execution Language for Web Services (BPEL4WS or BPEL for short) [1], WS-Coordination (WS-C) [24] and WS-Transaction (WS-T) [25]. BPEL is a workflow definition language that describes sophisticated business processes that can orchestrate Web Services. WS-Coordination and WS-Transaction complement BPEL to provide mechanisms for defining specific standard protocols to be used by transaction processing systems, workflow systems, or other applications that wish to coordinate multiple Web Services. These three specifications work in tandem to address the business workflow issues implicated in connecting and executing a number of Web Services that may run on disparate platforms across organizations involved in business scenarios.

The Business Process Execution Language for Web Services is the fusion of IBM'S WSFL and Microsoft's XLANG and it is actually supported by both. So far, it represents the most accredited candidate for becoming a future standard in the field of Web Services Choreography. For this reason it deserves to be studied and considered as a touchstone for any further effort in this field. BPEL allows for a mixture of block and graph-structured process models, thus making the language expressive at the price of being complex. Although BPEL is the most accredited proposal, it is remarkable how much attention it received, while more fundamental issues like expressiveness and adequacy have not been addressed. Another problem is that, although some attempts of formalizing a subset of BPEL have been performed [23], this language and also other similar proposals do not yet have any clearly defined official semantics. This because its complexity makes it difficult to formally define this framework, thus limiting the formal reasoning about the designed applications. For this reason, in this work we want to formally address the problem of defining workflows composing and coordinating Web Services. In particular, we propose a basic language to deal with Choreography. The aim of this language is to provide a mean to express common Web Service requirements.

	Completeness	Compositionality	Explicit Model of Parallelism	Explicit Model of Resources	Explicit Model of Error Handling
Turing Machine	☑	☒	☒	☑	☒
Lambda Calculus	☑	☑	☒	☒	☒
Petri Nets	☑	☒	☑	☑	☒
CCS	☑	☑	☑	☒	☒
Pi-Calculus	☑	☑	☑	☑	☒

Figure 1. Programming models synopsis

Relying on orchestration/choreography languages is argued to support the development of complex services in a more coherent and robust way [14,19] simplifying their analysis and design. Here we introduce in general terms the requirements a Choreography language for Web Services should meet. Business process choreography requirements involve asynchronous interactions, flow coordination, business transaction activity and management:

(i) Basic Flow Patterns:
- Sequence
- Conditional
- Parallel

(ii) Send/Receive to/from other WS

(iii) Send/Receive mapped on typed ports

(iv) Invocation of WS

(v) Error/Transaction Handling (Exceptions, Compensations)

In order to formally deal with these requirements we chose to start from the π-calculus [18], a well known process algebra which has been widely studied during the last fifteen years. Fig.1 motivates our choice comparing many different models from the point of view of many interesting features. The table shows the π-calculus as the most suitable choice for our purpose. Unfortunately, as emphasized by the last column, even the π-calculus does not support any transactional mechanism.

Since the Web Services environment requires that several Web Service operations have transactional properties and be treated as a single logical unit of work performed as part of a business transaction, in this paper we shall extend the basic calculus in such a way to include transactional facilities. Some other works have been presented in the past addressing similar issues. Anyway, all the past works committed only to ACID or Long-running semantics without providing a general framework for formalizing both the semantics. Instead, our attempt could be interpreted in this direction.

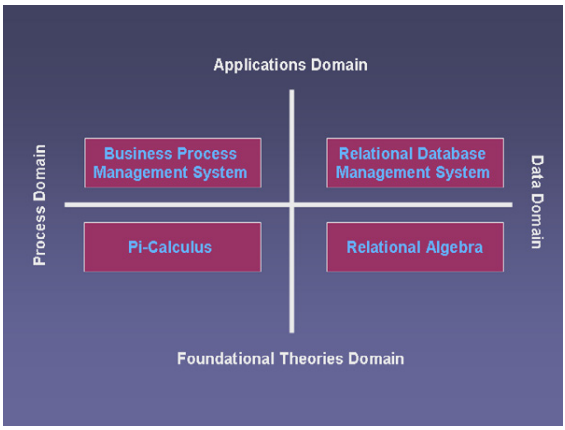

Figure 2. π-calculus as a foundation for Business Process Management System compared with relational algebra as a foundation for Relational Databases

Consider that, as the π-calculus is a natural way to express concurrency and as the most areas of programming are now concurrent — starting from microprogramming and device drivers level, passing from the GUI level and arriving to systems integration and business to business orchestration — its market could span a very wide range. For example, a practical application is represented by the XLANG Scheduler in Microsoft BizTalk Server [15], a recent tool used to integrate business systems. XLANG — the internal orchestration language of BizTalk — is explicitly built on a model from the π-calculus for a rigorous mathematical basis. Subsequently, also the definition of BPEL has been strongly influenced by this calculus.

The strong correlation between a theoretical and academical area and a business oriented one should not appear surprising as many people think. This is because a part of the people involved in the second one have been previously involved in the first one and, at the moment of choosing a valid paradigm for a Choreography language, they decided for an already deeply experimented one as, for example, the π-calculus. This appears completely natural as for the invention of the car: the more natural way for implementing such a mean of transportation was to enhance an already experimented one as the coach. Installing an engine on a basin would have resulted in a queer experiment, although pretty funny!

Because of this business interest in the π-calculus, we want to try to imagine a scenario in which this theory can be used as a foundation for Business Process Management System in the same way as relation algebra has been used as a foundation for relational databases. Such a synopsis is shown graphically in Fig.2.

3 The Event Calculus

The complexity of BPEL makes it difficult to formally define this framework, thus limiting the formal reasoning about the designed applications. In this paper we enhance our past work [13] – the Event Calculus — based on the idea of event notification as the only error handling mechanism. In that work we advocated that three different mechanisms for error handling are not necessary and we formalized a novel

choreography language. Here we are going beyond saying that such a mechanism is sufficient for modelling a wide range of issues related to Web Services Coordination in general. For doing this we extend the calculus with a multicast event notification mechanism more suitable for complex coordination scenarios. For instance, in e-commerce/business applications which involve more partners, certain events can be of interest for many of them (consider, e.g., the application managing the registration to a conference and the room reservation which exploits two Web services for supplying these tasks, the event "the credit card A is expired" should be notified to the hotel reservation service as well as to the conference registration service).

We are confident that a mechanism of multicast event notification represents the best choice for Web services coordination, especially in the context of e-business in which transactions play an important role. Our claim is supported also by different works recently proposed by other researchers who are using similar mechanisms (see for example the extensions of the CORBA transactional system reported in [7]).

More technically, the novelty we introduce in the Event Calculus regards: i) the introduction of multicast event notification, and ii) event scope names binding. The first extension allows an easier specification of complex coordination scenarios (such as e-commerce applications) with respect to the algebra we presented in our past work which was focused mainly on error handling mechanisms unification. The second extension also allows many new interesting behaviours which can be very useful in business scenarios. As previously discussed, the major barrier for a wider adoption of e-commerce is about security and privacy. We underlined the importance of the support offered by the underlying infrastructure for targeting these issues. The introduction of private event scope names that can be used also to deal with security and privacy. Private event scope names, indeed, allow us to manage events that can be observed only by a subset of processes involved in the application. In this way we guarantee that unauthorized information flows are not implemented by exploiting event notification. Consider, for example, the case in which an event means "User A used the credit card xxx to reserve a room in the hotel x"; this event should be used without authorization by a malicious process to monitor the usage of credit cards.

Finally, differently from previous work, the calculus we are going to describe allows for dynamic event scopes definition, that is to dynamically define the event an event scope is interested in.

3.1 Syntax

Let $\mathcal{N}$, ranged over by n, be the set of names and $\mathcal{T}$, ranged over by t, be the set of scope names. In the following, we use $u,v,\ldots$ to range over $\mathcal{N} \cup \mathcal{T}$, and $\tilde{u},\tilde{v},\ldots$ to denote lists of elements in $\mathcal{N} \cup \mathcal{T}$. The set of processes is defined by the following

grammar:

$$
\begin{array}{rll}
P, Q, R ::= & \mathbf{0} & \texttt{Normal Termination} \\
| & \overline{x}\ \tilde{v}.P & \texttt{Output} \\
| & x(\tilde{u}).P & \texttt{Input} \\
| & (u)P & \texttt{Private Name} \\
| & P \mid P & \texttt{Parallel Execution} \\
| & A(\tilde{u}) & \texttt{Process Invocation} \\
| & \texttt{signal}(t) & \texttt{Raising of a Signal} \\
| & [P, Q]_t & \texttt{Event Scope}
\end{array}
$$

We define free names $fn(P)$ of a process P as in the π-calculus with the necessary extension for $\texttt{signal}(t)$ and $[P, Q]_t$:
$fn([P, Q]_t) = (fn(P) \cup fn(Q)) \cup \{t\}$
$fn(\texttt{signal}(t)) = \{t\}$.

We are assuming a set of process constants, ranged over by A, in order to support process definition, whose definition follows:

Definition 3.1 [Process Definition] A defining equation for a process identifier A is of the form

$$A(\tilde{u}) \stackrel{def}{=} P$$

where it holds $fn(P) \subseteq \{\tilde{u}\}$ and $\tilde{u}$ is composed by pairwise distinct names

The first five operators are as usual: the $\mathbf{0}$ simply describes the normal termination of a process. The meaning of an *Output* $\overline{x}\ \tilde{v}.P$ is sending a list $\tilde{v}$, the *object* of the communication, through the channel x, the *subject*. The *Input* prefix $x(\tilde{u}).P$ represents the reception of the object $\tilde{u}$ through the channel x and it is a binder for the names $\tilde{u} \in \mathcal{N} \cup \mathcal{T}$ (these names can be channel names or scope names). The *New Name Creation* operator is also a binder for the name $n \in \mathcal{N} \cup \mathcal{T}$. The parallel operator represents the support for concurrency as the `flow` activity in BPEL. As in BPEL, the world here is modelled by concurrent activities which interact by message passing and event raising. BPEL allows for Web Services composition providing the `invoke` activity. In the same way, the process invocation à la π-calculus allows us to compose many different uncoupled services. So far the language is strictly similar to the π-calculus, it differs only for the last two operators. The first one is $\texttt{signal}(t)$ which produces a signal directed to all the event scopes identified by t. The second one is the definition of an event scope $[P, Q]_t$. Informally, the event scope $[P, Q]_t$ defines a process P to be run during the normal execution and an event handler Q associated with the event t (differently from the previous work, many event scopes can be interested to the same event, i.e. use the same identifier t). When a process in the system raises a $\texttt{signal}(t)$, the event handlers of all the event scopes interested to t that are ready to react will be eventually executed (the activation

is asynchronous due to physical latency) and the relative bodies (i.e. the processes managing the normal execution) are terminated. It follows that an event scope can catch a signal only once. Signals directed to nonexistent identifiers are lost.

Finally, it is worth to note that we can also dynamically define the event an event scope in interested in. Consider, for instance, the process $x(t).[P,Q]_t$ where the event scope name t is obtained as input on the channel x.

3.2 The Language Semantics

Now we shall give the semantics for the language in two steps, following the approach of Milner [17]. This approach consists in separating the laws which govern the static relations between processes from the laws which rule their interactions. We shall achieve this defining firstly a static Structural Congruence relation over syntactic processes. A Structural Congruence relation for processes is introduced as a small collection of axioms that allow minor manipulation on the processes structure. This relation is intended to express some intrinsic meanings of the operators, for example the fact that parallel is commutative. Secondly, we shall define the way in which processes evolve dynamically by means of a Labelled Transition System. Doing in this way we simplify the statement of the transition system just adding the (CONGR) rule in Table 1 which closes the transition relation under process order manipulation induced by Structural Congruence.

Definition 3.2 [Structural Congruence] The structural congruence on processes $\equiv$ is the smallest equivalence relation satisfying the followings and closed with respect to α-renaming, parallel composition and restriction:

(i) $(\mathcal{P}\ ,\ |\ ,\ \mathbf{0})$ is an Abelian Monoid:

$$\begin{array}{lcll} P_1|P_2 & \equiv & P_2|P_1 & \text{Commutativity} \\ (P_1|P_2)|P_3 & \equiv & P_1|(P_2|P_3) & \text{Associativity} \\ P|\,\mathbf{0} & \equiv & P & \mathbf{0} \text{ is nil element} \end{array}$$

(ii) $(u)\mathbf{0} \equiv \mathbf{0}$

(iii) $(u)(v)P \equiv (v)(u)P$

(iv) $(u)(P_1|P_2) \equiv P_1|(u)P_2$ if $u \notin \mathrm{fn}(P_1)$

(v) $A(\tilde{v}) \equiv P\{\tilde{v}/\tilde{u}\}$ if $A(\tilde{u}) \stackrel{def}{=} P$

In this paper we are using the usual definition for substitution: $P\{\tilde{v}/\tilde{u}\}$ means the replacement, in the process P, of each occurrence of a name in the ordered sequence $\tilde{u}$ with the correspondent name in the ordered sequence $\tilde{v}$.

Sometimes the semantics of a system is defined in term of a reduction relation which can result more concise. Anyway, in this case we found a labelled transition system a more elegant way for describing raising of signals and inter-scope interactions. Thus we decided to express the semantics in this way although it lacks of

brevity. The transition relations over system states are labelled by the *actions*. We have five kind of actions as defined in the following:

Definition 3.3 [Actions] The actions are given by

$$\alpha ::= \overline{x}\ \tilde{v} \mid x(\tilde{u}) \mid \langle t \rangle \mid t \mid \tau$$

We shall write Act for the set of actions.

The first action is sending the tuple $\tilde{v}$ via the channel x while the second is receiving the tuple $\tilde{u}$ via x. The third and the fourth ones stand respectively for the signalling of an event directed to the transactions identified by t and the notification of interest in catching the signal t. Finally, τ represents an internal action. We omit the definition for $fn(\alpha)$, $bn(\alpha)$ and $subj(\alpha)$. They represent, for actions, respectively the set of free names, bound names and names occurring as subject in a communication. These definitions are as usual with a straightforward extension for the signal labels.

Definition 3.4 [Transition Relations] The transition relations $\{\xrightarrow{\alpha} | \alpha \in Act\}$ on states $\mathcal{S}$ are defined by the rules in Table 1 where $P \xrightarrow{\alpha} P'$ means that the process P evolves in P' with the action α.

Table 1 is basically an optimized version of the one presented in our previous work with the addition of the multicast event notification mechanism and of the dynamic event scope definition. In order to express the semantic of the notification mechanism we follow the approach proposed in [4].

The prefix primitives for output and input on channels are described by axioms (OUT) and (IN), respectively, while (SIGNAL) shows that when the signal is performed it terminates. Rule (COM) shows the behavior of the communication between two processes and rule (PAR) describes the behavior of processes running in parallel (note that the rule does not allow to perform signal as well as event catching without considering all the processes running in parallel). Rules (RES) and (CONGR) are the classic ones used in the Pi-calculus for describing name restriction and the replaceability of processes with others structurally equivalent. Rule (SCOPE) shows that the event scope $[P, Q]_t$ evolves according to the behavior of P except in the case a signal or the reaction to the notification of the event t is performed. In particular, if the event t is notified by the process P (AUTO-RAISING) or t is notified by another process running in parallel (REACT) then the event scope behaves as Q. It is worth noting that if an event scope contains an event scope interested at the same event, say t, then solely the compensation associated to the outer event scope interested to t is performed. Rules (EVENT MULTICAST 1) and (EVENT MULTICAST 2) describe that, if the event t is signalled, it must be notified to all the processes interested in catching this event, while (EVENT CATCHING 1) and (EVENT CATCHING 2) impose that all the event scopes interested in t must react to this signal.

(Out) $\overline{x}\ \tilde{v}.P \xrightarrow{\overline{x}\ \tilde{v}} P$

(In) $x(\tilde{u}).P \xrightarrow{x(\tilde{u})} P$

(Signal) $\texttt{signal}(t) \xrightarrow{\langle t \rangle} \mathbf{0}$

(Com) $$\frac{P \xrightarrow{\overline{x}\ \tilde{v}} P' \qquad Q \xrightarrow{x(\tilde{u})} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'\{\tilde{v}/\tilde{u}\}}$$

(Par) $$\frac{P' \xrightarrow{\alpha} P''}{P' \mid P \xrightarrow{\alpha} P'' \mid P} \quad bn(\alpha) \cap fn(P) = \emptyset \quad \alpha \neq t, < t >$$

(Res) $$\frac{P \xrightarrow{\alpha} P'}{(u)P \xrightarrow{\alpha} (u)P'} \quad u \notin subj(\alpha)$$

(Congr) $$\frac{P \equiv P' \qquad P' \xrightarrow{\alpha} P'' \qquad P'' \equiv P'''}{P \xrightarrow{\alpha} P'''}$$

(Scope) $$\frac{P \xrightarrow{\alpha} P'}{[P,Q]_t \xrightarrow{\alpha} [P',Q]_t} \quad \alpha \neq \langle t \rangle, t$$

(Autoraising) $$\frac{P \xrightarrow{\langle t \rangle} P'}{[P,Q]_t \xrightarrow{\langle t \rangle} Q}$$

(React) $[P,Q]_t \xrightarrow{t} Q$

(Event Multicast 1) $$\frac{P \xrightarrow{\langle t \rangle} P' \qquad Q \xrightarrow{t} Q'}{P \mid Q \xrightarrow{\langle t \rangle} P' \mid Q'}$$

(Event Multicast 2) $$\frac{P \xrightarrow{\langle t \rangle} P' \qquad Q \xrightarrow{t\!\!\!/}}{P \mid Q \xrightarrow{\langle t \rangle} P' \mid Q}$$

(Event Catching 1) $$\frac{P \xrightarrow{t} P' \qquad Q \xrightarrow{t} Q'}{P \mid Q \xrightarrow{t} P' \mid Q'}$$

(Event Catching 2) $$\frac{P \xrightarrow{t} P' \qquad Q \xrightarrow{t\!\!\!/}}{P \mid Q \xrightarrow{t} P' \mid Q}$$

Table 1
Labelled Transition System

4 An E-commerce Scenario

In this section an e-commerce scenario will be presented in order to show the potentiality of the Event Calculus for describing long running transactions and business activities. As we said, long running transactions can involve other transactions. For this reason, generally, a coordinator is needed in order to handle the state of the long running transaction and to activate compensations when some inner transactions fail. Referring to [9] we consider the example of a customer application which

allows to purchase a new set of formalwear items, a suit, a tie and a pair of shoes from a shopping portal. The customer application requires that all the items must be purchased otherwise the order must be cancelled. The shopping portal could be seen as an interface between the customer application and a coordinator which manages the Business Activity (BA) linked to the purchase order. In particular let's suppose that the three items are sourced by three different suppliers with no trust relationship between them. Each supplier has a public service which can be invoked by the coordinator in order to start three different internal Business Activities (let's name *BA1* the Business Activity for buying the suit, *BA2* the Business Activity for buying the tie and *BA3* the Business Activity for buying the pair of shoes). In this case three different transactions will be performed. When each of them completes with success a `completion` message is sent to the coordinator. The coordinator can send a message of completion to the application customer, through the shopping portal interface, when all the internal transactions have been completed with success. Now let's consider the fact that *BA1*, *BA2* and *BA3* are invoked concurrently by the coordinator and each of them can be terminated with a completion or with a failure. A completion corresponds to the purchase of an item, otherwise the failure represents the fact that the supplier cannot source the item. For instance it is possible that *BA1* and *BA3* complete buying the suit and the pair of shoes, and *BA2* fails because the supplier cannot source the tie. In this case a message of failure is sent to the coordinator which could try to obtain the tie from another supplier (let's name *BA4* the alternative Business Activity for buying the tie). If this is possible, a new internal transaction can be activated between the coordinator and the new supplier's service in order to complete *BA4*, otherwise *BA1* and *BA3* have to be compensated. The compensation means that the coordinator has to cancel the prior completed transactions with the supplier of the suit and with the supplier of the shoes. In this case, after the compensations, the coordinator is in a state semantically equivalent to the state before the purchase order operations were carried out. The shopping portal knows the state of the coordinator and can signal to customer application the fact that the order cannot be completed.

For the sake of simplicity we assume that the business application BA (the shopping portal) we are going to model exploits four business activities, namely *BA1*, *BA2*, *BA3* and *BA4* where *BA4* is the only alternative for *BA2* (while for *BA1* and *BA3* there is not alternative). We first introduce the channel names, event scope names and processes will be used to describe the activity and their corresponding meaning. Let $invokeS1$, $invokeS2$, $invokeS3$, $invokeS4$, $receiveS1$, $receiveS2$, $receiveS3$ and $receiveS4$ be the channels used to invoke (resp. receive the response) the Web service supplying business activity *BA1*, *BA2*, *BA3* and *BA4*, respectively. Let *receive* and *reply* be the channels used by the service supplier (the shopping portal) to receive service invocations (we use *req* to denote the request parameters) and to reply to the customer application, respectively. Let $completion1$, $completion2$, $completion3$ and $completion4$ be the event names used to denote the completion of business activities *BA1*, *BA2*, *BA3* and *BA4*, respectively. Let *coo* be the event which represents the failure of the long running transaction managed by

BA, *abort* be the event name denoting that the involved activities (*BA1*, *BA2*, *BA3* and *BA4*) should abort. We use $ABORTHANDLER1$, $ABORTHANDLER2$, $ABORTHANDLER3$, and $ABORTHANDLER4$ to denote the processes which manage this task for business activity *BA1*, *BA2*, *BA3* and *BA4*, respectively. Let *notok* be the event name representing that all the activities completed in a successful way must be cancelled (by executing the compensation processes). We use $CANC1$, $CANC2$, $CANC3$, and $CANC4$ to denote the processes which manage this task for business activity *BA1*, *BA2*, *BA3* and *BA4*, respectively. Let m be the event representing that the business activity *BA2* has failed and then the alternative *BA4* is to be considered.

For the sake of simplicity, the model we present exploits the conditional operator [1] . $IsresOK?P : Q$ that tests if res (which is the return value of a Web Service invocation) is equal to *ok* (which means the business activity completes with success) then the process P is performed, in the opposite case Q is executed.

The definition of the business activity BA, expressed by process $PShop$, follows.

$$
\begin{aligned}
BA1 \ ::= & \ [\overline{invokeS1}(req).receiveS1(res).IsresOK \ ? \\
& \ \texttt{signal}(completion1) : \\
& \ \texttt{signal}(coo), ABORTHANDLER1]_{abort} \\
& \ | \ [\mathbf{0}, [\texttt{signal}(count), CANC1]_{notok}]_{completion1}
\end{aligned}
$$

$$
\begin{aligned}
BA2 \ ::= & \ [\overline{invokeS2}(req).receiveS2(res).IsresOK \ ? \\
& \ \texttt{signal}(completion2) : \\
& \ \texttt{signal}(m), ABORTHANDLER2]_{abort} \\
& \ | \ [\mathbf{0}, [\texttt{signal}(count), CANC2]_{notok}]_{completion2}
\end{aligned}
$$

$$
\begin{aligned}
BA3 \ ::= & \ [\overline{invokeS3}(req).receiveS3(res).IsresOK \ ? \\
& \ \texttt{signal}(completion3) : \\
& \ \texttt{signal}(coo), ABORTHANDLER3]_{abort} \\
& \ | \ [\mathbf{0}, [\texttt{signal}(count), CANC3]_{notok}]_{completion3}
\end{aligned}
$$

$$
\begin{aligned}
BA4 \ ::= & \ [\overline{invokeS4}(req).receiveS4(res).IsresOK \ ? \\
& \ \texttt{signal}(completion4) : \\
& \ \texttt{signal}(coo), ABORTHANDLER4]_{abort} \\
& \ | \ [\mathbf{0}, [\texttt{signal}(count), CANC4]_{notok}]_{completion4}
\end{aligned}
$$

$$
\begin{aligned}
Comp \ ::= & \ \texttt{signal}(abort) \\
& \ | \ \texttt{signal}(notok) \\
& \ | \ \overline{reply}(BAnotok)
\end{aligned}
$$

$$Coord \ ::= BA1 \mid [BA2, BA4]_m \mid BA3 \mid [\mathbf{0}, Comp]_{coo}$$

$$Counter \ ::= [\mathbf{0}, [\mathbf{0}, [\mathbf{0}, \overline{reply}(BAok).\texttt{signal}(req)]_{count}]_{count}]_{count}$$

$$PShop \ ::= receive(req).([Coord \mid Counter, \mathbf{0}]_{req})$$

Process $PShop$ is the process supplying the shopping portal, it replies to the consumer $BAok$ if the long running transaction is completed or $BAnotok$ in the

[1] This construct can be easily encoded in our calculus (see [22])

case of failure (and in this case it manages the compensation). It is worth noting that the process $Counter$ implements a counter by exploiting the event $count$, when the event $count$ is notified three times it notifies the event req (which means that all the business activities have been completed) and then the process $PShop$ can terminate by replying $BAok$.

5 Conclusions

In this paper we enhanced our past work - the Event Calculus - which was based on the idea of event notification as the only error handling mechanism in Web Services Choreography. While the previous work were addressed on the problem of defining a sort of kernel language for modelling Web Service orchestration and error handling mechanisms, here we have extended the language introducing two main novelties: i) a multicast event notification mechanism, and ii) an event scope names binding mechanism.

The first extension allows an easier specification of complex coordination scenarios (such as e-commerce applications) with respect to the algebra we presented in our past work which was focused mainly on error handling mechanisms unification. We want to add some considerations about the notification mechanism: when an event is signalled, no operations are performed until the system has activated all the event scopes interested to that event. In the case of distributed Web Services and event scopes, in order to model in a more realistic way the mechanism it can be interesting to assume that scopes catch the event in an asynchronous manner. On the other hand, the signal mechanism already allows to express that the notification of events is asynchronous. The second extension also allows many new interesting behaviours which can be very useful in business scenarios, such as the opportunity of handling security and privacy issues or the dynamic event scope definition for managing multiple instances of the same application. This feature shares some similarities with the technique proposed by BPEL for the same purpose. Although we mainly presented e-commerce applications for our language, we consider that such a calculus represents a foundational framework able to deal with any aspect of Web Services Coordination.

We consider also that the proposed language shares some features with BPEL, StAC [5] and π_t-calculus [3]: they can all be viewed at the same level of programming abstraction. As future work we intend to investigate the expressiveness of such proposals. In particular, we feel that the technique proposed by BPEL to manage multiple instances of the application (based on correlation-sets) has some similarities with the way we propose for defining at runtime the event a scope is interested to, for example as in $x(t).[P, Q]_t$. A more detailed and formal comparison between such languages and the calculus here presented is left as future work.

A last remark is about the need for timed transactions. Presently, we believe that a notion of time in long running transactions can be useful in business scenarios (refer to [10]). Other researchers consider that the notion of time should be introduced both at the model level and at the protocols and implementation levels.

XLANG itself, for example, contains a notion of Timed Transaction as a special case of long running activity.

References

[1] Tony Andrews, Francisco Curbera et al. - *Businnes Process Execution Language for Web Services Specification, Version 1.1*, 5 May 2003.

[2] A. Arkin - *Business Process Modelling Language (BPML) specification*, BPMI, http://www.bpmi.org, June 2002.

[3] L.Bocchi, C.Laneve, G.Zavattaro - *A Calculus for Long Running Transactions*. FMOODS '03, Paris, December 2003. LNCS.

[4] N. Busi, R. Gorrieri, G. Zavattaro - *Process Calculi for Coordination: from Linda to JavaSpaces*. In Proc. of International Conference on Algebraic Methodology and Software Technology (AMAST'00), 198-212 LNCS 1816, Springer-Verlag, 2000.

[5] Michael Butler, Carla Ferreira - *An Operational Semantics for StAC, a langage for Modelling Long-running Businness Transactions*. COORDINATION 2004.

[6] E.Christenses, F.Curbera, G.Meredith, S.Weerawarana. *Web Services Description Language (WSDL 1.1)* http://www.w3.org/TR/wsdl, W3C, Note 15, 2001.

[7] Iain Houston, Mark C. Little, Ian Robinson, Santosh K. Shrivastava, Stuart M. Wheater - *The CORBA Activity Service Framework for supporting extended transactions*. Softw., Pract. Exper. 33(4): 351-373 (2003).

[8] R. Khalaf, S. Tai, and S. Weerawarana - *Web services, the next step: A framework for robust service composition*, CACM, Special Issue on Service-Oriented Computing, M.P. Papazoglou and D. Georgakopoulos, Eds., October 2003.

[9] Introducing WS-Transaction, An Arjuna Technologies Report - http://www.arjuna.com

[10] Cosimo Laneve and Gianluigi Zavattaro. Foundations of web transactions. In *Fossacs'05*, volume 3441 of *LNCS*. Springer-Verlag, 2005.

[11] F.Leymann - *Web Services Flow Language (WSFL 1.0)*- http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[12] F. Leymann and D. Roller - *A quick overview of BPEL4WS*, IBM DeveloperWorks, August 2002 - http://www-106.ibm.com/developerworks/

[13] M.Mazzara, R.Lucchi - *A Framework for Generic Error Handling in Business Processes*. First International Workshop on Web Services and Formal Methods (WS-FM), 2004.

[14] L.G. Meredith, Steve Bjorg - *Contracts and Types*. Communication of the ACM, October 2003

[15] Microsoft Corp. Biztalk Server. http://www.microsoft.com/biztalk

[16] Robin Milner - *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[17] R. Milner - *Function as Processes*. Mathematical Structures in Computer Science, 2(2):119-141, 1992.

[18] Robin Milner, Joachim Parrow, David Walker - *A Calculus for Mobile Processes*. Journal of Information and Computation, 100:1-77. Academic Press, 1992.

[19] Chris Peltz - *Web Services Orchestration and Choreography*. IEEE Computer October 2003 (Vol.36, No 10), pages 46-52.

[20] *SOAP - Simple Object Access Protocol*. http:// www.w3.org/TR/soap

[21] S.Thatte - *XLANG: Web Services for Business Process Design*, Microsoft Corporation, 2001 http:///www.gotdotnet.com/team/xml_wsspecs/xlang-c

[22] David N. Turner - *The Polymorphic Pi-calculus: Theory and Implementation*. PhD thesis, University of Edinburgh, 1995.

[23] Mirko Viroli - *Towards a Formal Foundation to Orchestration Languages*. First International Workshop on Web Services and Formal Methods (WS-FM), 2004.

[24] WS-Coordination Specification
http://www-128.ibm.com/developerworks/library/specification/ws-tx/

[25] WS-Transaction Specification
http://www-128.ibm.com/developerworks/library/specification/ws-tx/

[26] *XML - Extensible Markup Language 1.0*
http://www.w3.org/TR/2000/REC-xml-20001006