CESSA

Compositional Evolution of Secure Services using Aspects

ANR project no. 09-SEGI-002-01

Model and formal architecture specification

Abstract. In this deliverable we present the formal service model that allows service-oriented architectures to be formally specified and the semantics of service compositions as well as individual services to be formally defined.

We present work related to the formal treatment of service-oriented architectures, define the syntax and formal semantics of our service model, provide a comprehensive illustrating example by showing how the BPEL orchestration language can be represented in our model, and give an outlook how the service model is to be extended by a model for the specification of (correctness and security) properties as well as a model for the evolution of service compositions using aspects.

Deliverable No.	D1.2
Task No.	1
Type	Deliverable
Dissemination	Public
Status	Final
Version	2.0
Date	10 January 2011
Authors	D. Allam, R. Douence, H. Grall, I. Mejía, J.-C. Royer, M. Südholt (EMNantes); M. S. Idrees, Y. Roudier (Eurecom); Anderson Santana De Oliveira (SAP)

Contents

Chapter 1

Introduction

The CESSA project aims at providing new methods and techniques for secure service-oriented applications. Its partners mainly pursue support for the preservation of security properties in the presence of evolution for applications that involve large-scale server infrastructures and resource-limited, *e.g.,* mobile, devices. A sound formal basis for the definition of properties, their analysis and enforcement in service-oriented architectures is a necessary requirement for this endeavor.

The formal treatment of service-based systems has received quite some attention over the last decade. However, the corresponding body of approaches does not meet the needs of the CESSA project, most importantly because formal models for service choreography are still in their infancy and the (numerous) models for service orchestration frequently do not cover important features of mainstream orchestration languages, such as BPEL.

In this report, we introduce a formal framework for service-oriented computing, suitable for service choreography and orchestration. As described in Deliverable D1.1 ([11], Sec. 2.1), the framework is structured into two layers: a collaboration layer and a process layer. The framework can be instantiated in different ways, each instantiation providing a language for describing processes. We propose an instantiation, based on the π-calculus [29], and designed to model the orchestration language BPEL [33]. The formalization of BPEL is a real challenge since this language contains different features, like scopes, variables, process instances, correlations or compensations. It therefore provides a good test to assess our framework.

This deliverable is structured as follows. In chapter 2 we present related work on the formal definition of service compositions through choreography and orchestration. The formal framework is defined in chapter 3. In chapter 4 we present a first application of the framework by formally defining an essential part of the BPEL workflow language. Chapter 5, finally, brings forth tracks for the extension of the service framework by a model for collaboration-global properties and another model for the evolution of services using aspects, both of which are to be defined in the following phase of the CESSA project.

Chapter 2

State of the Art of Orchestration and Choreography

The interplay between orchestration and choreography is complex and should be clarified. For some authors it is two different standards to define behaviour of web-services. The distinction between orchestration and choreography is subtle, [12] states they should both merge in the future. Bruni, in [5], says that the distinction is blurred but orchestration is mainly concerned with control, communications and data flow between services while choreography focuses on the interactions between several services. Orchestration defines the local view of service interactions while choreography addresses a global view of these interactions. The orchestration describes the interactions of one service with other services. An orchestration makes explicit a central coordinator which is responsible for invoking and combining the web-services realising the composition. A choreography does not assume a central coordinator, it defines the collaboration as the set of messages and the interaction rules between the services. As in other component models, for instance [34, 13], two formal sub languages are of interest. The first one is about control, communication, data of a single process and the second one is about the description of synchronisations and interactions of a set of processes. Both languages are not independent, they are two faces of a formal component model, one for the internal description of each process, and one for the composition and observation of processes. The global semantics of a composite component must mix both languages in order to fully describe the system behaviour.

2.1 Formal Models of WS Orchestration

A wide variety of formal models for web services and their orchestration exists [4, 6, 22, 40, 28] or more generally models for Service Oriented Computing (SOC) [21, 23, 24, 39, 36, 19, 5]. There are few surveys about these formal models, the most exhaustive is [38]. It presents various approaches relying on finite state machines, Petri nets, abstract state machine, process algebras and the B language. However, these surveys do not really compare the benefits and drawbacks of these models.

One of the earlier approaches was to use finite state machines and specifically Petri nets.

Surveys about the Petri net are [27, 26]. [26] analyses the translation of workflow languages into Petri nets. A specific point is made about BPEL and the authors point out some difficulties with the fault handler and the dead-path elimination. A more precise comparison of the two feature complete Petri nets approaches for BPEL is done in [27]. The finite state machines do not allow a flexible form of mobility. These formalisms are useful regarding the rich set of models, verification techniques and tools. However, some approaches like Petri nets do not fully support composition which seems a real benefit of web-service.

Several approaches rely on process algebras, mainly CCS like in [6]. Process algebras are really adequate formalisms for BPEL, for instance sending or receiving message can be expressed directly, the `flow` construction is a pure parallel composition of processes, etc. CCS provides a native synchronous call, no computation and no timeout, and it does not provide a simple way to model mobility. It is more laboured to do asynchronous communications and to express data computations. LOTOS [25] can be used as process algebra since it provides data handling and several operators for synchronisation and event processing. In [14] the author presents a framework for the design and the verification of WSs using LOTOS and its tools. It specifies a two-way mapping between abstract specifications written using LOTOS and executable Web services written in BPEL4WS. The translation includes compensation, event, and fault handlers. Another interest of this work is that various verifications are made possible using the CADP tool box. One step further could be to rely on the E-LOTOS standard which in addition allows the modelling of time. However, for the BPEL formalisation, it is desirable to have a form of mobility, which is not primitive in classic process algebras like CCS or LOTOS. One way is to allow channel to be communicated to partners, thus many approaches exploit the π-calculus as [28, 21, 5]. Orc is another way [36, 19] which has an original model calculus based on four combinators.

The notion of `LINK` is a way to define an explicit ordering of processes during a parallel composition. This feature is really unusual in process algebras and it is generally too difficult to specify it. For instance [6, 28] did not consider it. A more declarative approach is better. Process algebras rely on process composition and the execution rules are responsible of the exact process ordering. The reader can find in [23] a formal description of this flow graph construction. [40] provides a simple semantics of LINK, and it quotes [20] as a related work covering this aspect.

Another important aspect is correlation set. Correlation can be used on every messaging activity: `receive`, `reply`, `onMessage`, `onEvent` and `invoke`. A web service describes a type of service (or a process type) and several instances of this service can be concurrently running to serve different requests. Correlation sets are the mechanism to ensure that replies are routed to the right client. They are kinds of message filters which can select data on a channel to restrict communication between two partners. This aspect has often been defined using the more abstract notion of session, that is a kind of private interaction is established between the two partners. [5] argues that "Though correlation sets offer a good expressiveness, we argue that they may complicate static analysis, because all interactions rely on data values". The notion of session seems a more convenient abstraction mechanism for arbitrarily enclosing interactions between pairs. However, from a point of view of security it seems better to formalise precisely the notion of correlation set, as for instance done by Viroli in [40]. The notion of session is more developed for choreography models, like session types calculi, see Section 2.2.

Some calculus have introduced a notion of transaction to model long running interactions which can be down or subject to various errors. The notion of transaction is important in web-based services, it is different from the data-oriented transaction used in DBMS [35]. The classic ACID properties are not valid. In web-services short atomic transactions are possible but the novelty is the notion of business activity. They are process-oriented and long duration transactions over the web. In this setting atomicity is called semantic and supported by compensation mechanism ; isolation is global and the control flow is explicit. For instance, [28] considers an operator which run as a global process but it can be interrupted by an event and in this case the compensate activity is run. This is like the disable operator from the LOTOS language [25]. We consider that such a mechanism can be implemented with more basic mechanisms, namely messages and a state machine based control.

The article [28] did also a fine analysis of the various errors, faults and exceptions mechanisms provided by BPEL. BPEL provides three mechanisms related to abnormal situations: exception, event and compensation handling. Event is a classic concept associated with event handler and the `pick` construction which allows a choice on an event occurrence. The two other mechanisms are only provided in the `scope` construction. Fault handlers are also classic in programming languages, however their activity is parallel with the main activity in the scope. Compensation handlers are associated with the termination of an activity to provide a way to manage long running transactions. [28] presents a π-calculus semantic for web services with a precise analysis of these three mechanisms. In fact the transaction mechanism introduced in the π-calculus extension is able to encode the various event and exception behaviours in a scope. This transaction operator is based on the receipt of an event. One conclusion of [28] is that the event mechanism, completed with usual parallel constructions, can simulate fault and compensation handlers of BPEL.

The networking topology of web services can be viewed as a set of communication links from server nodes to client nodes. We should distinguish servers and clients and furnish a simple way to define the topology which is a general graph. Two way interactions can be established by first a communication from the client to the server and then one in the opposite direction provided that the client has sent in its messages a channel for the reply.

Sequence is a composition construction which should deserve our attention. Sequence in BPEL have a special semantics which is not easily captured by the sequence construction of process algebras. Canal et al in [6] defines it as a set of parallel process which are explicitly synchronise on termination, but they do not manage data in their formalisation. [28] defines the sequence as a suite a processes with a set of variable information which are transmitted from the first to the last during the processing steps. Another interesting approach, is to have a pipeline construction as in Orc [19] or CaSPiS [5]. In Orc the `F >x> G` is the sequential combinator and its informal meaning is: Each value published by `F` initiates a parallel activity of `G` with `x` bound to this value. There are as many `G` parallel instances as published values by `F`. CaSPiS uses a different, yet more general, pipeline operator than Orc. `P > Q` means to feed `Q` with all the values produced by `P`. In Orc the pipeline operator needs an explicit variable name, thus it is equivalent to `P > (?x)Q` in CaSPiS. Viroli in [40] proposes a more classic way to model it based on a concept of store which captures imperative assignments on a set of variables.

We expect to define various controls on the obtained specifications. Thus we think valuable

to have a typed language, more precisely to have types for messages and for channels. The type of a channel can be viewed as an operation signature in more traditional programming languages. We could also consider some specific kinds of channels, used as for instance for events, fault or process termination. An advantage of this approach is for instance to verify the syntax of communications or to check that data circulating on channels are compliant with the specifications. There are already some attempts in this area, for instance type-checking was studied in [22]. [40] proposes several simple controls which guarantee uniqueness of correlation assignments, and absence of deadlocks for well-formed processes.

Termination of processes is important in BPEL, it is used in the semantics of the compensation in a scope. The process termination seems a feature useful in BPEL as it was in the LOTOS language [25]. In this language there is a separated and explicit semantics for the description of process termination. This aspect was also specified in other process algebras, like in Orc. We note that it can be considered as a particular event which can lead to some simplification if we use explicit events.

2.1.1 Correlation Set and Spawning of Process Instances

Internet services are created and removed dynamically. New partners register and others disappear. Communication links or network may be subject to various failure or dynamic changes. In a loosely coupled world such as web services, there are no direct references between the interacting processes. Web services use partner links to identify potential partners. In fact they identify a port type which can be viewed as an interface. Of course, services of the same kind share a same interface and many different services can have the same port type. This mechanism allows to configure the set of interacting services later in the development process, at run time, and even to dynamically reconfigure these partners. It is a convenient way but with the drawback that a service cannot statically predict which are its exact partners. Thus partners identification should rely on the data exchanged during communications. When a web service interacts with others it does not known the precise identity of its partners, the correlation set mechanism is the key to relate partners of a web service. For instance, we may have a web service which needs to communicate several times with the same partner but at different and unrelated instants. In this case, the correlation set should be initiated and it is defined as a part of messages exchanged between partners. It makes it possible to rout messages to the correct process instance.

A correlation can be initiated or not thanks to the `initiate` attribute which can be yes, join or no. The join case is used when the specifier wants to initiate it if not already initiated. A correlation set can be initiated at most only once. A message can contain several parts used to correlate messages, a message can carry multiple correlation sets. These correlation sets can also be nested if they are inside the scope construction. Quoting the BPELV2.0 specification [33] page 77: "When correlation set in a message does not match an already initiated correlation set in the process instance or if the correlation set is not initiated, the message MUST not be delivered to an IMA[1]".

Another peculiarity of WS-BPEL is that processes are rather process type descriptions. A run-

[1] Inbound Message Activity

ning process instance is dynamically spawned at message receipt and if the property `createInstance` was set to true. A correlation set is specific to a given process instance. A process instance defines an interacting session with partners using the same correlation set. But one additional complication are the contraints governing multiple starting activities, see the BPELspecification [33] page 90.

In [40] a formal semantics of correlation sets is defined. It shows that it can be considered as a variable which can be set only once (or a late bound constant). It can be useful if one partner has to interact in several parts of the session and with different partners. As in [40] if we introduce a store, correlation key can be viewed as a restrict case of variable, once set their value cannot change.

The notion of originator/follower is also important. The initiator is the partner which sends the first message of a session defining a correlation. It is responsible for spawning the process instance managing the session. Followers are other partners involved in the same session and using the same correlation.

2.1.2 Three Formal Models

This subsection explores some existing formal models to target one reference model for our future semantics.

Amongst the candidate models we consider [40, 28, 3].

[28] introduces an extension of the π-calculus with a notion of transaction. Although this mechanism can be defined in π-calculus it is better to consider it as a native construction of the language. The semantics of BPEL is formally translated into this core language. This translation can be simplified by considering only asynchronous invocation and receipt of a message. There is no handling of data and global variables, only messages circulating on the channels as in π-calculus. This work is mainly devoted to the semantics of the scope and the mechanisms of event handlers, fault handlers and compensation.

Viroli [40] introduces also a π-calculus based semantics but focusing on correlation sets. It details the session management in BPEL. "The way this is achieved is by splitting the whole business process into different isolated working sessions called process instances, each having the same behaviour of the business process specification but running on a different scope with a different store of variables and pertaining different subsequent interactions. The correlation mechanism is introduced precisely to specify in a declarative way how different interactions can be identified as belonging to the same process instance, and in particular, how incoming messages are to be routed to the proper process instance. To this end, a particular kind of variable called property is introduced in BPEL. It is basically a late-bound constant once initialised its content will never change and is defined so as to alias the single field of one or more message types." [40] The core language introduces a notion of imperative stores that enables to manage variables and correlation sets. The semantics is obtained using a labelled transition system as target and operational semantic rules. One originality of the core language is the instance spawning noted $[\pi : S]R$ which represents the orchestration R, the correlation set π and the S spawned instance. In traditional π-calculus this mechanism is generally modelled by the use of infinite replication (noted !) and restriction. The language is extended with algorithmic constructions, specially the

process sequence construction. However, aspects related to events, faults and compensation are not described.

Another notable work is the one from Abouzaid and Mullins [1, 2, 3]. The author proposes a formal BP-language close to the π-calculus and inspired from the work of Lucchi and Mazzara for handlers and of Vitori for correlation sets. The BP-language is a π-calculus with a replication, an if then else and a guarded choice constructions. The scope is modelled by the restriction and the handlers execute concurrently to the scope. The specification of the various handlers (event, fault, termination and compensation) is done in a similar way than [28] but without a specific transaction construct. The semantics of the correlation sets use variables and a spawning construct as in [40].

One interesting application of this work is the use of HAL [1], a tool dedicated to the verification of π-calculus, to check the BP formulae and to prove or disprove properties. Another point is the translation from the BP formulae to BEPL expressions, the two languages are rather close and the translation can be automated. However, the overall rules set described in [3] is rather complex since there are 14 congruence rules and 23 reduction rules.

2.2 Formal Models for Choreography

Since the domain of service choreographies, in contrast to service orchestrations, is still in its infancy, few researchers have considered formal models for service choreography, even for the most well-known among these models, the web services choreography description language (WS-CDL) [42].

In the following we present several representative approaches in this domain, notably session types that have been developed independently from service choreography but have been applied to that domain and have several interesting characteristics (property support through static typing, global and local choreography specifications, flexible communication model ...). We intend to extend the formal model presented in this report with support for aspects and session types in order to govern the evolution of service compositions. Finally, we briefly describe several other approaches to the formal definition of session types that use different style of semantics and suggest different ways to ensure composition properties over service choreographies.

2.2.1 Session Types

Session types denote an approach to the definition of communication protocols between distributed entities. A static type system is used to enforce fundamental correctness properties of the resulting communicating systems.

Session types have garnered much attention over the last five years, in particular, because calculi have been developed for the precise definition of interactions in distributed systems, notably object-oriented distributed systems [15]. While being developed independently of service choreographies, they have been proposed as a formal foundation of the international choreography standard, WS-CDL [7].

Session types have the following characteristics:

C1 **Global protocol.** Session types define a representation of the global interactions between all services involved in a service composition. The resulting global protocol facilitates understanding of the system and can be used to analyze and verify its formal properties.

C2 **End-point projection.** The global protocol defined by session types can be projected onto communication end points, that is, essentially onto individual sites in a distributed system. Type checking of the choreography can thus be defined exclusively in local terms.

C3 **Flexible communication model.** While initially developed for synchronous interactions, as frequently used in object-oriented and service-oriented systems, session types also accommodate asynchronous [17] and event-based distributed systems [18]. Moreover, session types also support multi-party interactions [17], in contrast to many to most other approaches to the formal definition of interactions in terms of protocols.

C4 **Static property enforcement.** The types ensure different interaction properties and the type system enables their static enforcement.

2.2.2 Models specific to web services choreography

The models specifically developed for web services choreography typically define (small) language that captures a (small) subset of the WS-CDL specification and provide a formal semantics.Some approaches focus on how correctness properties of service choreographies can be ensured.

The overall number of published approaches is still very small (less than 20 articles in international conferences). Hence, it does not make sense to categorize the existing approaches, *e.g.,* according to the semantic paradigm they are based (operational, denotational ...) or the properties (compatibility, fault handling, ...) they allow to tackle. In the following we therefore briefly present a recent approach that uses different underlying semantics and addresses different properties.

Yang *et al.* [43] present the *CDL approach* that provides a small language with explicit representations for basic CDL activities, in particular, conditions, repetitions, work units, external and internal choice, and parallel execution. The specificity of this approach to WS-CDL is obvious, *e.g.,* because it represents work units explicitly in the syntax, which associate a condition guard, a repetition guard and a nested activity. The semantics is defined in terms of a small-step operational semantics and supports manual reasoning as well as reasoning using model checking after translation to a model. In both cases, no assurance of the correctness of the reasoning processes are provided (*e.g.,* no relation between model generation process and the semantics is defined).

In 2010, the CDL approach has been extended in order to accommodate exception handling and finalisation in WS-CDL [44]. Activities are extended by exception throwing activities and a so-called "chaos" activity that represents, among others, non-terminating choreographies. The authors define a corresponding denotational semantics but no operational interpretation or property support is provided.

Chapter 3

The Formal Model

We briefly recall the service model that we want to formalize. It has been described in the first deliverable of Task 1 [11, Sect. 2.1].

The service model is based on three layers, corresponding to collaborations, processes and services. Each process is under the control of a peer. Processes can collaborate, by exchanging messages. A collaboration can involve different peers, when the messages exchanged cross peer boundaries. A message corresponds to the request of a service or to the subsequent reply. Services are provided by server processes and required by client processes. Precisely, a process can invoke a service provided by another process by sending a request: it therefore requires the presence of the service to execute.

3.1 The Collaboration Model

At the collaboration level, there is a set of processes that interact each other via message boxes. We assume that each process is located at some location and that at each location there is a unique process, with an input message box and an output message box. A location can therefore be considered as an identifier for a process. In the following, we do not model peers. Actually, a peer can simply be modeled as a set of locations, corresponding to the processes under the control of the peer.

3.1.1 Syntax for Collaborations

Table 3.1 describes the syntax used to describe collaborations. A collaboration is simply a finite set of located processes.

3.1.2 Reduction Semantics for Collaborations

Here is the semantics of the collaboration calculus. It depends on a reduction relation defined for processes. Precisely, assume $([l : p_l])_l$ is a collaboration. A distributed configuration contains two parts, a multiset θ of messages $l!m(v)$ migrating between processes and corresponding to the

Collaboration	$c ::= \emptyset$	Empty Collaboration
	$\mid [l : p]$	Process p at Location l
	$\mid c, c$	Collaborations in Parallel
Location	$l \in \mathcal{L} \subseteq \mathcal{V}$	Subset of Values

Table 3.1: Collaboration Calculus

$$\frac{\beta_k \rightarrow \beta'_k}{\theta, \left([l : \beta_l]\right)_{l \neq k}, [k : \beta_k] \rightarrow \theta, \left([l : \beta_l]\right)_{l \neq k}, [k : \beta'_k]} \text{[LOC]}$$

$$\theta, [k : I_k, O_k + k'!m(v), P_k], \left([l : \beta_l]\right)_{l \neq k} \rightarrow \theta + k'!m(v), [k : I_k, O_k, P_k], \left([l : \beta_l]\right)_{l \neq k} \quad \text{[OUT]}$$

$$\theta + k!m(v), [k : I_k, O_k, P_k], \left([l : \beta_l]\right)_{l \neq k} \rightarrow \theta, [k : I_k + m(v), O_k, P_k], \left([l : \beta_l]\right)_{l \neq k} \quad \text{[IN]}$$

Table 3.2: Collaboration Semantics

network and a map $([l : \beta_l])_l$ assigning to each location l a local configuration β_l. A message $l!m(v)$ corresponds to the call of operation m with argument v. The operation m is provided by the process located at l. Each local configuration β_l is a triplet I_l, O_l, P_l, where I_l is an input message box, O_l is an output message box and P_l is a process in its environment. The message boxes are multisets of messages. There is a union operation over multisets: if m and m' are multisets, then $m + m'$ is equal to the union of m and m'. We assume that a reduction relation is defined over local configurations: $\beta_l \rightarrow \beta'_l$ means that local configuration β_l reduces (in one step) in local configuration β'_l.

The reduction relation for a collaboration is defined between distributed configurations and is an extension of the reduction relation defined over local configurations for individual processes. See Table 3.2 for details.

- [LOC]: local reduction of a process
- [OUT]: message emission from a process to the network
- [IN]: message reception from the network to a process

Note that the semantics of the process language can be kept abstract. Our model is a framework that can lead to different instantiations: each instantiation provides a language for describing processes.

3.2 The Process Language

We now come to the process level. In order to model the orchestration language BPEL, following a now common trend, we use a process language, a variant of the π-calculus. Two main strategies could be used to model a language like BPEL.

Encoding. The source language can be translated into the target language, the π-calculus in its original form. The translation corresponds to an encoding of the different constructs present in BPEL by using the small set of constructs present in the π-calculus.

Advantage: the target language is a small and formal language with a well-founded theory.

Drawback: the encoding entails a gap between the definition of the process at the source level and its definition at the target level.

Extension. The target language, the π-calculus, is extended in order to ease the translation from the orchestration language BPEL.

Advantage: the encoding is made easy, with only a minor gap between the source and target processes.

Drawback: the target language is more complex, with theoretical foundations to build.

We have chosen to follow the extension strategy, while trying to minimize the drawback by selecting standard constructs with a clear semantic interpretation.

3.2.1 Syntax for Processes

The process language, defined in Table 3.3, has the following features.

- The basic processes are the empty process, which does nothing, and two elementary actions, assignment and message sending. Assignment allows variables to be updated, while message sending allows messages to be sent either to the output message box for external communication, or to the input message box for internal communication.
- The guarded sum allows message receiving with multiple choices. Guarded processes wait for input messages. When an input message becomes available in the input message box, a process waiting for this kind of message can be fired.
- Three control operators are present: sequence, parallel and alternative.
- Recursive definitions are possible, with a dedicated construct.
- Variables are introduced by a local definition, with a given scope.

Here are the main variations with respect to the π-calculus.

- The language has state variables and expressions to denote the values assigned to the variables. Each state variable has a scope and an initial value, and can then be updated by an assignment. In the π-calculus, there are no state variables, but they can be encoded.

- Expressions are used to denote not only variable contents, but also boolean conditions in alternatives (with values `true` and `false`) and locations in message sending. They form a higher-order algebra, built from values, either constants or monadic functions. In the π-calculus, there are only channels.
- A channel is represented here by an ordered pair, a location and a name for the message. Locations can be computed from expressions, allowing a weak form of mobility. An expression denoting a location computes a location belonging to a fixed set of locations. Contrary to the π-calculus, there is no way to define a new location, since locations are determined by the network at the collaboration level. There is also a particular location, representing the current host and denoted by `h`: it is used for internal communication. As in the π-calculus, channels are monadic: they convey a single argument.
- Input prefixes are not binders. Instead, any message reception involves a variable assignment.
- Non-determinism is limited to processes with input prefixes. In the π-calculus, there is a general sum operator.
- In the π-calculus, there is no sequence and no alternative[1], since they can be encoded. However, the standard encoding heavily uses the new operator, allowing new channels to be generated, but this operator is not present in our language.

3.2.2 Congruence Rules

In the process language, there are two binders: the scope operator for state variables and the recursion operator for process variables. Here are the algebraic laws satisfied by the processes. They generate a congruence relation between processes.

1. Two Processes are congruent if they only differ by a change of bound variables.
2. The n-adic sum operator Σ is commutative and associative, with $\emptyset$ as neutral element.
3. The parallel operator is commutative and associative, with $\emptyset$ as neutral element.
4. The sequence operator is associative, with $\emptyset$ as neutral element.
5. A recursive definition is congruent to its expansion.
$$\mu X.p \equiv p[\mu X.p/X]$$
6. A scope applied to the null process is the null process.
$$\texttt{let}\ x = v\ \texttt{in}\ \emptyset \equiv \emptyset$$
7. The declaration order of variables in a scope does not matter.
$$\texttt{let}\ x = v\ \texttt{in}\ \texttt{let}\ y = u\ \texttt{in}\ p \equiv \texttt{let}\ y = u\ \texttt{in}\ \texttt{let}\ x = v\ \texttt{in}\ p \quad (x \neq y)$$

[1] Actually, there is often a specific alternative allowing channels to be compared.

Process	$p ::= \emptyset$	Null process
	$\mid a$	Action
	$\mid \Sigma_{i \in I} m_i(x_i).p_i$	Guarded Sum
	$\mid p \,;\, p$	Sequence
	$\mid p \parallel p$	Parallel
	$\mid e \,?\, p \,:\, p$	Alternative
	$\mid \mu X.p$	Recursive Process
	$\mid X$	Process Variable
	$\mid \texttt{let}\ x = v\ \texttt{in}\ p$	Scope
Action	$a ::= x = e$	Assignment
	$\mid e!m(e)$	Output
Expression	$e ::= x$	State Variable
	$\mid v$	Value
	$\mid e(e)$	Application
Message Name	$m \in \mathcal{M}$	Finite Set
Value	$u, v \in \mathcal{V}$	Set of Values

Table 3.3: Process Calculus

3.2.3 Reduction Semantics for Processes

The reduction semantics defines a reduction relation between process configurations, as described in Table 3.4. A process configuration $I,\ O,\ \Gamma \vdash\ p$ is defined as follows.

- I: multiset of messages $m(v)$, called input message box
- O: multiset of messages $l!m(v)$, called output message box
- Γ: environment, a stack of bindings $(x \mapsto v)$
- p: active process

We assume that expressions can be evaluated in environments. The value assigned to expression e in environment Γ is denoted by $e[\Gamma]$. An environment can be extended and updated. Let Γ be an environment. Let x be a variable, and v a value. If the binding $(x \mapsto v)$ belongs to environment Γ, then it can be updated as follows: the environment $\Gamma.(x \mapsto v')$ represents the environment where the binding $(x \mapsto v)$ has been replaced with the binding $(x \mapsto v')$. The environment can also be extended: the environment $\Gamma + (x \mapsto v)$ is equal to the environment Γ extended with the binding $(x \mapsto v)$ at the top of the stack. Here is the meaning of the different rules.

$$I + m_j(v),\ O,\ \Gamma \vdash\ \Sigma_{i\in I} m_i(x_i).p_i \ \rightarrow\ I,\ O,\ \Gamma.(x_j \mapsto v) \vdash\ p_j \quad (j \in I) \quad [\texttt{IN}]$$

$$I,\ O,\ \Gamma \vdash\ x = e \ \rightarrow\ I,\ O,\ \Gamma.(x \mapsto e[\Gamma]) \vdash\ \emptyset \quad [\texttt{VAR}]$$

$$I,\ O,\ \Gamma \vdash\ e_1!m(e_2) \ \rightarrow\ I,\ O + e_1[\Gamma]!m(e_2[\Gamma]),\ \Gamma \vdash\ \emptyset \quad (e_1[\Gamma] \neq \texttt{h}) \quad [\texttt{EXT}-\texttt{OUT}]$$

$$I,\ O,\ \Gamma \vdash\ e_1!m(e_2) \ \rightarrow\ I + m(e_2[\Gamma]),\ O,\ \Gamma \vdash\ \emptyset \quad (e_1[\Gamma] = \texttt{h}) \quad [\texttt{INT}-\texttt{OUT}]$$

$$\frac{I,\ O,\ \Gamma \vdash\ p\ \rightarrow\ I',\ O',\ \Gamma' \vdash\ p'}{I,\ O,\ \Gamma \vdash\ p\,;\,q\ \rightarrow\ I',\ O',\ \Gamma' \vdash\ p'\,;\,q}\ [\texttt{SEQ}]$$

$$\frac{I,\ O,\ \Gamma \vdash\ p\ \rightarrow\ I',\ O',\ \Gamma' \vdash\ p'}{I,\ O,\ \Gamma \vdash\ p \parallel q\ \rightarrow\ I',\ O',\ \Gamma' \vdash\ p' \parallel q}\ [\texttt{PARA}]$$

$$\frac{e[\Gamma] = \texttt{true}}{I,\ O,\ \Gamma \vdash\ e\,?\,p\ :\ q\ \rightarrow\ I,\ O,\ \Gamma \vdash\ p}\ [\texttt{IF}+] \qquad \frac{e[\Gamma] = \texttt{false}}{I,\ O,\ \Gamma \vdash\ e\,?\,p\ :\ q\ \rightarrow\ I,\ O,\ \Gamma \vdash\ q}\ [\texttt{IF}-]$$

$$\frac{I,\ O,\ \Gamma + (x \mapsto v) \vdash\ p\ \rightarrow\ I',\ O',\ \Gamma' + (x \mapsto v') \vdash\ p'}{I,\ O,\ \Gamma \vdash\ \texttt{let}\ x = v\ \texttt{in}\ p\ \rightarrow\ I',\ O',\ \Gamma' \vdash\ \texttt{let}\ x = v'\ \texttt{in}\ p'}\ [\texttt{LET}]$$

Table 3.4: Reduction Semantics

- [IN]: a process can consume a message in the input message box.
- [VAR]: the content of a variable can be updated.
- [EXT − OUT]: a process can send a message to another process via the output message box.
- [INT − OUT]: a process can send a message to itself via the input message box.
- [SEQ]: a process followed by another process can reduce.
- [PARA]: a process in parallel with another process can reduce.
- [IF+]: an alternative reduces to the first process if the condition is satisfied.
- [IF−]: an alternative reduces to the second process if the condition is not satisfied.
- [LET]: a process inside a scope can reduce.

3.2.4 Correlation

Correlation is used to select the process receiving an input message. Given a value v conveyed by an input message $m(v)$, assume that different processes $m(x_i).p_i$ (for i in some finite set I) can

respond to this message:

$$I + m(v),\ O,\ \Gamma \vdash\ m(x_i).p_i\ \rightarrow\ I,\ O,\ \Gamma.(x_i \mapsto v) \vdash\ p_i.$$

(Generally, the guarded process $m(x_i).p_i$ will belong to a guarded sum $\Sigma_{j\in J} m_j(x_j).p_j$.) In the standard semantics, the process fired is randomly chosen. With correlation, it is possible to select a subset of the processes according to the value of the message. To each variable x_i, a correlation pre-order[2] R_i is assigned, allowing values to be compared. Typically, the pre-order R_i is induced by a correlation function $g_i : \mathcal{V} \rightarrow \mathcal{V}$ and an associated partial order $\leq_i$ over $g_i(\mathcal{V})$. Indeed the pre-order R_i over $\mathcal{V}$ can then be defined as follows:

$$v\ R_i\ v' \stackrel{\text{def}}{\Leftrightarrow} g_i(v) \leq_i g_i(v').$$

The principle for correlation is simple: a guarded process $m(x_i).p_i$ can be selected if

$$\Gamma(x_i)\ R_i\ v.$$

This property means that the variable x_i is related to the value since its correlation content, computed with g_i, is "included" in the correlation content of the value. Consider as examples two limit cases. First, assign to each variable a constant function. Then for any i, the pre-order R_i is equal to the universal relation $\mathcal{V} \times \mathcal{V}$. Therefore any process waiting for the message can be selected. This is the standard semantics described in the preceding subsection. Second, order the set $\mathcal{V}$ of values by the identity relation (the diagonal). Then correlation corresponds to an exact matching between the content of the variable and the input value.

Note that after the communication, the correlation content $g_i(\Gamma(x_i))$ of the variable x_i is still present but possibly may have increased. Often, it is useful to specify how the correlation content evolves. This evolution constraint can be represented by a second pre-order R'_i included in R_i. That is the reason why the guard of a reception process also declares a pre-order:

$$m_i(x_i : R'_i).p_i.$$

Likewise, the assignment operator declares a pre-order:

$$(x : R'_i = e).p.$$

For instance, given a variable x with correlation pre-order R, in order to forbid the modification of its content, just use the identity relation. In order to require an initialization of its content, use the strict pre-order associated to R.

The syntax of the process language with correlation is defined in Table 3.5. Here is an account of the extension.

- `let` $x : R = v$ `in` p

  When a variable x is introduced with a scope, a pre-order R is assigned to the variable. Typically, the pre-order is induced by a correlation function g and a partial order $\leq_g$ over $g(\mathcal{V})$:

  $$v\ R\ v' \Leftrightarrow g(v) \leq_g g(v').$$

  It represents the way the content of the variable can evolve.

[2] A pre-order is a reflexive and transitive relation.

Process	$p ::= \emptyset$	Null process
	$\mid a$	Action
	$\mid \Sigma_{i \in I} m_i(x_i : R).p_i$	Guarded Sum
	$\mid p \,;\, p$	Sequence
	$\mid p \parallel p$	Parallel
	$\mid e \,?\, p \,:\, p$	Alternative
	$\mid \mu X.p$	Recursive Process
	$\mid X$	Process Variable
	$\mid \texttt{let}\ x : R = v\ \texttt{in}\ p$	Scope
Action	$a ::= x : R = e$	Assignment
	$\mid e!m(e)$	Output
Expression	$e ::= x$	Variable
	$\mid v$	Value
	$\mid e(e)$	Application
Message Name	$m \in \mathcal{M}$	Finite Set
Value	$u, v \in \mathcal{V}$	Set of Values
Correlation Pre-Order	$R \subseteq \mathcal{V} \times \mathcal{V}$	Reflexive and Transitive Relation

Table 3.5: Process Calculus - With Correlation

- $m(x : R).p$

  A guarded process specifies a pre-order for the variable receiving the message. This pre-order is a subset of the pre-order assigned to the variable. It determines the way the content of the variable can evolve during the reception.

- $x : R = e$

  An assignment specifies a pre-order for the variable assigned. This pre-order is a subset of the pre-order assigned to the variable. It determines the way the content of the variable can evolve during the assignment.

The semantics of the process language is defined in Table 3.6. Process configurations are enriched in order to keep track of the pre-orders associated to variables. A configuration I, O, γ, $\Gamma \vdash p$ is now defined as follows.

- I: multiset of messages $m(v)$, called input message box
- O: multiset of messages $l!m(v)$, called output message box

$$\frac{\Gamma(x_j)\ R_j\ v}{I + m_j(v),\ O,\ \gamma,\ \Gamma \vdash\ \Sigma_{i\in I} m_i(x_i : R_i).p_i \to I,\ O,\ \gamma,\ \Gamma.(x_j \mapsto v) \vdash\ p_j}\ (j \in I, R_j \subseteq\ \gamma(x_j)) \quad [\texttt{IN}]$$

$$\frac{\Gamma(x)\ R\ e[\Gamma]}{I,\ O,\ \gamma,\ \Gamma \vdash\ x : R = e \to I,\ O,\ \gamma,\ \Gamma.(x \mapsto e[\Gamma]) \vdash\ \emptyset}\ (R \subseteq\ \gamma(x)) \quad [\texttt{VAR}]$$

$$I,\ O,\ \gamma,\ \Gamma \vdash\ e_1!m(e_2) \to I,\ O + e_1[\Gamma]!m(e_2[\Gamma]),\ \gamma,\ \Gamma \vdash\ \emptyset \quad (e_1[\Gamma] \neq \texttt{h}) \quad [\texttt{EXT}-\texttt{OUT}]$$

$$I,\ O,\ \gamma,\ \Gamma \vdash\ e_1!m(e_2) \to I + m(e_2[\Gamma]),\ O,\ \gamma,\ \Gamma \vdash\ \emptyset \quad (e_1[\Gamma] = \texttt{h}) \quad [\texttt{INT}-\texttt{OUT}]$$

$$\frac{I,\ O,\ \gamma,\ \Gamma \vdash\ p\ \to\ I',\ O',\ \gamma,\ \Gamma' \vdash\ p'}{I,\ O,\ \gamma,\ \Gamma \vdash\ p\,;q\ \to\ I',\ O',\ \gamma,\ \Gamma' \vdash\ p'\,;q}\ [\texttt{SEQ}]$$

$$\frac{I,\ O,\ \gamma,\ \Gamma \vdash\ p\ \to\ I',\ O',\ \gamma,\ \Gamma' \vdash\ p'}{I,\ O,\ \gamma,\ \Gamma \vdash\ p \parallel q\ \to\ I',\ O',\ \gamma,\ \Gamma' \vdash\ p' \parallel q}\ [\texttt{PARA}]$$

$$\frac{e[\Gamma] = \texttt{true}}{I,\ O,\ \gamma,\ \Gamma \vdash\ e\,?\,p\ :\ q\ \to\ I,\ O,\ \gamma,\ \Gamma \vdash\ p}\ [\texttt{IF}+] \qquad \frac{e[\Gamma] = \texttt{false}}{I,\ O,\ \gamma,\ \Gamma \vdash\ e\,?\,p\ :\ q\ \to\ I,\ O,\ \gamma,\ \Gamma \vdash\ q}\ [\texttt{IF}-]$$

$$\frac{I,\ O,\ \gamma + (x \mapsto R),\ \Gamma + (x \mapsto v) \vdash\ p\ \to\ I',\ O',\ \gamma + (x \mapsto R),\ \Gamma' + (x \mapsto v') \vdash\ p'}{I,\ O,\ \gamma,\ \Gamma \vdash\ \texttt{let}\ x : R = v\ \texttt{in}\ p\ \to\ I',\ O',\ \gamma,\ \Gamma' \vdash\ \texttt{let}\ x : R = v'\ \texttt{in}\ p'}\ [\texttt{LET}]$$

Table 3.6: Reduction Semantics – With Correlation

- γ: correlation environment, a stack of bindings $(x \mapsto R)$ mapping variables to pre-orders
- Γ: environment, a stack of bindings $(x \mapsto v)$
- p: active process

The correlation environment is initialized via the rule [LET], which allows a process inside a scope to reduce. Here are the semantic rules modified by correlation.

- [IN]: a process can consume a message in the input message box when the content of the variable receiving the message evolves following the pre-order specified.
- [VAR]: the content of a variable can be updated when the content of the variable evolves following the pre-order specified.

Chapter 4

Application to "Business Process Execution Language"

In the chapter, we use our formal model to represent processes defined in the "Business Process Execution Language" (BPEL). First, we discuss the features of BPEL that are covered by the formal model. Second, we deal with a concrete example.

4.1 BPEL Formalization

This subsection discusses the features of BPEL covered by the previous formal model. We also identify some aspects which need more work or have been discarded. The model is a process algebra with mailboxes and integrate an orchestration part. It can be seen as a π-calculus but with characteristics of the λ-calculus with imperative variables. Thus it is at first sight more complete and abstract than BPEL. It is more abstract since it can express location computation, thus a form of mobility. However, there are many syntactic details we discarded for instance, the XML representation and XPath manipulations, standard elements, the naming of processes, the type information, etc. Our current calculus do not provide activity extension and is mainly devoted to the specification of executable BPEL processes.

The process calculus have native constructions for the following features:

- `empty` : The null process is noted $\emptyset$.
- `invoke/receive` : Call a web service/receive an invocation. We only consider asynchronous communications, synchronous and other variants can be easily encoded in our calculus. There are messages which are sent and stored in process mailboxes. The notation $loc!m(e)$ stands for sending a message m with parameters e to the location loc, this is equivalent to an invocation.
- The mailboxes did not specify a specific policy to manage messages. In BPEL a process wait for receive but the specification in [33] does not specify a message delivery mechanism.

- Associated with message invocation and receipt activities is the so-called correlation set mechanism. Our calculus provides an abstract view of this mechanism. We have functions which compute the correlation parts associated to variables and messages, then communications rules cope with checking the correlation set. The mechanism supports multiple correlations and nested correlations.

- `assign` : Variable assignment, this is managed by the $=$ action.

- `sequence` : Sequential ordering of activities but in an imperative setting, native rules have been defined for that.

- `if` : We make explicit this construction in our calculus.

- `pick` : This is a non deterministic choice of a branch driven by the occurrence of an event, it is subsumed by the guarded sum Σ.

- `flow` : This is the parallel composition ($||$).

- `scope` : It defines a process activity, it is denoted by the *let* $x = v$ *in* p construction. Thus our variables are quite classic imperative variables with assignment and scope as in BPEL.

- We introduce an explicit process recursion, it is useful to spawn a process in case of a new correlation, and also to define various other control structures.

There are some constructions like `while`, `RepeatUntil`, and `ForEach` which are easy to encode in this calculus using conditional and recursive process definitions. Note that the `ForEach` construction can be sequential or parallel, the parallel case can be encoded with the parallel and the recursion constructions. However, some features are not yet covered for various reasons or are more difficult to encode. There are various internal events (fault, exception, timeout, termination). Currently internal communications can be defined by a communication from a process to its proper mailbox (the h keyword). We do not manage timeout but it could be done with some specific process counting time and sending the corresponding timeout. The `wait` action can be simulated in a similar manner.

The `exit` terminates all the activities related to the current BPEL process, it needs more work to be encoded.

Fault handlers are also classic in programming languages, however their activity is parallel with the main activity in the scope. We can define some specific events denoting faults and with the parallel construction fault handlers can be defined in our calculus.

We do not yet define a compensation mechanism, it relies on events and the scope mechanism. As it was done in [28, 1] we think that a similar way is possible in our calculus.

The notion of `LINK` is a way to define an explicit ordering of processes during a parallel composition. As said in the state of the art part this feature is really unusual in process algebras thus we skip it.

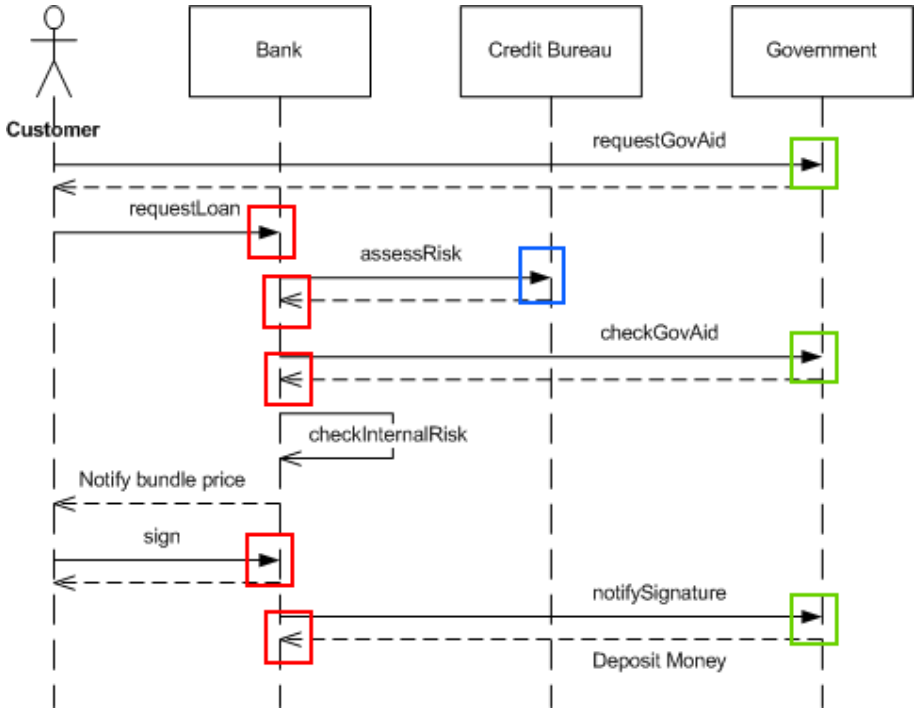


Figure 4.1: Multi domain logging evolution, [16]
ask for government aid and to make a loan request to the bank.

Captured-state tracked by Bank logging system

Captured-state tracked by Credit Bureau logging system

Captured-state tracked by Government logging system

4.2 Example: From BPEL to the Formal Model

In the section, we present a complete example using correlation sets in BPEL as defined in [8]. We will take the loan negotiation scenario described in [16]. In the figure 4.1, one can easily identify the different peers involved. The example involves different peers:

Customer: Who is able to ask for government aid and to make a loan request to the bank.

Bank: Who verifies that the person asking for the loan has satisfied all requirements given the amount of money and the concerned interest ratings.

Credit bureau: Who store and safeguard credit information of individuals and companies. Using these information, the credit bureau can help partners to make decisions on whether or not to grant credit, in terms of their own credit granting policies.

Government: Who check if a citizen is eligible to a government financial aid.

Our example will be presented differently in two parts:

Example 4.1: (Part 1: Getting the better loan)

Suppose that a client "A" wish to request a loan to buy a new house. Knowing that he is subscribed at two banks, Bank1 and Bank2, he wants to launch in parallel two requests, one for each bank, to chose the better offer proposed for the loan. That means that there will be two parallel processes executed at the client "A" side, each one is designated to communicate with the correspondent process at the bank side.

The global process of this loan process is :

- the client send his personal information to the government to have the right for a loan.
- the government reply by sending an approval and a client id
- the client must send the government approval and his id to bank1 and bank2 to ask for a loan
- the bank checks the client id and the approval with the government
- the bank checks the risk of the loan with the credit bureau and make a decision
- the bank reply by sending a contract to the client specifying the loan rate and interests
- the client sign the most convenient contract for his demand.

The question is: After the client has sent the two requests loan, once he receives a response from a bank, how does the client know which reply is for which loan request? The answer is : by using correlation sets.

The correlation sets are created as follows:

- As we explain before, the client must launch two parallel processes, each one is identified by a correlation set "`loanRequest(ClientID, OrderID)`", the "`ClientID`" is the id sent by the government and the "`OrderID`" is a value to distinguish between the two loan requests. So, when a bank replies by sending the contract, the reply message must contain the correlation set "`loanRequest`" to be assigned to the right process.
- Of course client "A" is not the only client who communicates with the bank for a loan, that's why the bank uses also the correlation set "`loanRequest(ClientID, OrderID)`", to distinguish between different clients.
- We need also another correlation set "`contract(ContratID)`" which will be initiated by the bank and sent to the client when the bank replies his customer. Then, when the bank sends the contract message, he must add two correlation sets "`loanRequest(ClientID, OrderID)`" and "`contract(ContratID)`". So on, if the client accepts the contract conditions, he will send a confirmation message containing the correlation set `contract`, without needing to send the correlation set loanRequest to reach the right process. To clarify the idea, at each peer side, we will have a process defining two correlation sets, "`loanRequest`

(ClientID, OrderID)" and "contract(ContractID)", such that, once values of these correlation sets are initiated, an incoming message containing values of one or both correlation sets can be assigned to the right process. The general principal for that in BPEL is: "when multiple correlation sets are used in an inbound message activity, knowing that all correlation sets are initiated, a message must match all such correlation sets to be delivered to the activity in the given process instance", [8].

In Listing 4.1, we present the BPEL client process interacting with the previous one.

```
1  <process name="RequestLoanProcess" ...>
2     <correlationSets>
3        <correlationSet name="loanRequest"
4           properties="cor:clientID cor:orderID" />
5        <correlationSet name="contract"
6           properties="cor:contractID" />
7     </correlationSets>
8      ...
9     <sequence>
10       <invoke partnerLink="government"
11          operation="requestAid"
12          inputVariable="info"
13          outputVariable="Aid">
14          ...
15       </invoke>
16       <invoke partnerLink="bank"
17          operation="requestLoan"
18          inputVariable="clientReq">
19
20          <correlations>
21             <correlation set="loanRequest" initiate="yes"/>
22          </correlations>
23       </invoke>
24
25       <receive partnerLink="bank"
26          operation="notify"
27          input="contractMsg">
28          <correlations>
29             <correlation set="loanRequest" initiate="no"/>
30             <correlation set="contract" initiate="yes"/>
31          </correlations>
32          ...
33       </receive>
34       <if name="contractMsgNotBlockedByGovernment">
35          ...
```

```
36          <invoke partnerLink="bank"
37                  operation="contractReply"
38                  inputVariable="ReplyContract">
39
40                  <correlations>
41                     <correlation set="contract" initiate="no"/>
42                  </correlations>
43                     ...
44                   <!-- process the complete signing
45                           contract depending on the client
46                           reply-->
47          </invoke>
48       </if>
49    </sequence>
50 </process>
```

Listing 4.1: BPEL example to request a loan

In Listing 4.2 we present the BPEL bank process. For the moment, we do not care about fault handlers in BPEL.

```
1  <process name="LoanProcess" ...>
2     <correlationSets>
3        <correlationSet name="loanRequest"
4           properties="cor:clientID cor:orderID" />
5        <correlationSet name="contract"
6           properties="cor:contractID" />
7     </correlationSets>
8       ...
9     <sequence>
10       <receive partnerLink="client"
11          operation="requestLoan"
12          inputVariable="clientReq"
13          createInstance="yes">
14
15          <correlations>
16             <correlation set="loanRequest" initiate="yes"/>
17          </correlations>
18       </receive>
19
20       <invoke partnerLink="government"
21          operation="checkGovernmentAid"
22          inputVariable="data"
23          output="confirmation">
```

```
24          ...
25          <if name="ifGovernmentConfirmTheAidForClient">
26             <condition>
27                confirmation = "true"
28             </condition>
29             <sequence>
30               <invoke partnerLink="creditBureau"
31                    operation = "checkRisk"
32                    inputVariable="loan" />
33               <receive partnerLink="creditBureau"
34                  operation="assessRisk"
35                  inputVariable="risk" />
36                  ...
37                <reply partnerLink="client"
38                    operation="notify"
39                    input="contractMsg">
40                    <correlations>
41                       <correlation set="loanRequest"
42                            initiate="no"/>
43                       <correlation set="contract"
44                             initiate="yes"/>
45                    </correlations>
46                    ...
47                </reply>
48
49                <receive partnerLink="client"
50                    operation="replyContract"
51                    inputVariable="clientReply">
52                    <correlations>
53                       <correlation set="contract"
54                            initiate="no"/>
55                    </correlations>
56                    ...
57                    <!-- process the complete signing contract
58                          depending on the client reply-->
59                 </receive>
60              </sequence>
61              <!-- if the government deny the aid -->
62              <else>
63                 <reply partnerLink="client"
64                      operation="notify"
65                      inputVariable="contractMsg">
66                      ...
```

```
67                          <!-- sending a contractMsg in
68                                  blocked situation caused by the
69                                   government -->
70                          <correlations>
71                             <correlation set="loanRequest"
72                                  initiate="no"/>
73                             <correlation set="contract"
74                                  initiate="yes"/>
75                          </correlations>
76                       </reply>
77                    </else>
78                 </if>
79           </invoke>
80       </sequence>
81 </process>
```

Listing 4.2: BPEL example to receive a loan request

In the following two examples we will use the notation :

$$\texttt{let}\ x = \texttt{e}\ \texttt{in}\ p$$

instead of :

$$\begin{array}{l}\texttt{let}\ x = \bot\ \texttt{in}\\ \quad (x = \texttt{e})\ ; p\end{array}$$

That will makes our examples easier to read and understand.

The BPEL example in Listing 4.1 can be translated in our proposed model as follows.

```
let info = clientInfoValue in
   (*The requestAid and sendAid services match with
   the first synchron invoke in the BPEL example*)
   govLoc!requestAid(info) ;
   sendAid(aid) ;
   let loanRequest = getID(aid) in
      let loan = amountValue in
         let clientReq = pair(loanRequest)(loan) in
            bankLoc!requestLoan(clientReq) ;
            let contractMsg : S1 = triplet(loanRequest)(⊥)(⊥) in
               notify(contractMsg : S2) ;
               getContractCondStatus(contractMsg)
               ?
               let reply = replyContentMessageValue in
                  let clientReply = pair(getContractID(contractMsg))(reply) in
                  bankLoc!replyContract(clientReply) ;
                  ...
               : ∅
```

The BPEL example in Listing 4.2 can be translated in our proposed model as follows.

```
μX.(let clientReq : R = ⊥ in
   requestLoan(clientReq : R1) ;
   let data = pair(getClientID(clientReq))(getAid(clientReq)) in
      govLoc!checkGovAid(data) ;
      let confirmation = ⊥ in
         replyAid(confirmation) ;
         confirmation
         ?
         CBLoc!checkRisk(getAid(clientReq))
         assessRisk(risk) ;
         ...
         let contract = new(⊥) in
            let contractMsg = triplet(getLoanReq(clientReq))(contract)(contractCond) in
            loc(clientReq)!notify(contractMsg) ;
            let clientReply : R2 = pair(contract)(⊥) in
               replyContract(clientReply : R3) ;
               ...
         :
         let contract = new(⊥) in
            let contractMsg = triplet(getLoanReq(clientReq))(contract)(contractCond) in
            loc(clientReq)!notify(contractMsg)) ‖ X
```

Matching between BPEL and our abstract model In the following, we will present how some BPEL syntactic constructs are translated in our model based on the example.

1. **Partner Links:** they are used to identify the parties that interact with the business process. Each link can be seen as an interface to interact with a peer, and on this interface we define operation that could be called by the other peer and also operations that the process can invoke at the partner.

   In our model, the notion of partner links can be reconstructed from the following notions:

   - each service proposed by a peer, is represented by a channel, so the peer who proposes the service listen on the corresponding channel for an incoming message, while the peer who wishes call the service send a message on this channel. In our example, we can see clearly this matching, each operation used in the BPEL examples 4.1 and 4.2 appears as a channel on the corresponding model.
   - a partner is seen as a location, we do not need to dissociate partners by different partnerLinks. The notion of locations in our model replaces also **Binding** used in BPEL.

2. **createInstance=”yes”:** This enables a receive activity to start a new instance of the BPEL process, that mean that we create a new environment for a new process. This environment will contain also a copy of all previous declaration defined by the initial BPEL process before the receive activity was launched.

   In our model, we can represent that by :

   $$\mu X.(\texttt{let}\ x = v\ \texttt{in}\ p \parallel X)$$

   as it was done for the loan process at the bank side. With this representation, we can create a new process instance as done in BPEL by putting the ”μX.” at the top of the process, so on, all declared variables will be copied in the new environment created for the new process.

3. **correlation Sets:** As we can clearly see in the two BPEL examples, 4.1 and 4.2, correlation set must be declared each time this correlation is a part of an input or an output variable with a parameter `"initiate"`. This parameter is putted at `yes` if the correlation set doesn't take previously a value for the current process, if the value is putted at `no` that means the process must not initiate the value of the correlation set in the process because it was initiated before.

   In our model, correlation sets are represented by pre-order relations used only at the reception of a new message, which is syntactically different from BPEL. To clarify the idea, in the BPEL example 4.1, the second activity in the sequence is an invoke of the operation ”`requestLoan`” which has an input variable ”`clientReq`”. As we see in this example, the correlation set ”`loanRequest`” was declared at this step with assigning the value ”yes” to the parameter ”`initiate`”, whether, in the translated model of this BPEL example, the process don't care about any correlation set:

$$\texttt{bankLoc!requestLoan(clientReq)}.$$

But at the bank reception side, we declare two pre-order relations R and R_1 which will have the same role of correlation sets in BPEL:

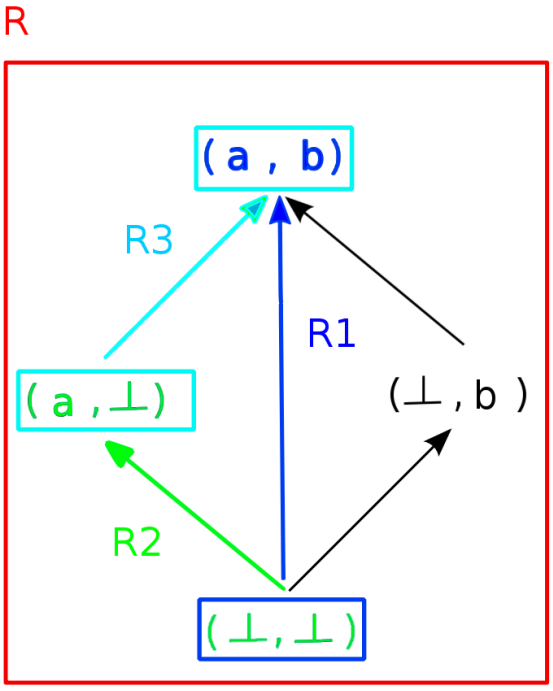
$$\texttt{let clientReq} : R = \perp \texttt{ in requestLoan(clientReq} : R_1)$$

Following, we will explain all pre-order relations used in the example to show how this syntax match with correlation sets used in BPEL.

- R **pre-order:** this relation is presented in the figure 4.2. In our example, this relation was decomposed into three sub pre-order relations, R_1, R_2 and R_3.
  - R_1 was used in the process loan at the bank side, at the reception of the client loan request. Based on this relation, the incoming message, "`clientReq`" must match with the following transition:
    $$(\perp, \perp) \rightarrow (a, b)$$
    This translation has the same meaning of putting "`initiate`" parameter of the correlation set "`loanRequest`" to "yes", because the received message will contain at first parameter the value of the correlation set "`loanRequest`" which will be assigned to $\perp$.
  - R_2 was used in the loan process at the bank side to assign a value to the variable "`clientReply`". This corresponds in BPEL to initiate the correlation set "`contract`" by putting the "`initiate`" parameter to "yes".
  - R_3 was used in the loan process at the bank side, at the reception of the message, clientReply. This relation match with putting the initiate parameter of the correlation set "`contract`" to "no".
- S **pre-order:** this relation is presented in the figure 4.3. In our example, this relation was decomposed into two sub relations S_1 and S_2.
  - S_1 was used at the loan request process at the client side, to assign a value to the variable "`contractMsg`". This relation corresponds to initiate a value to the correlation set "`loanRequest`" in the BPEL process by putting the "`initiate`" parameter at "yes".
  - S_2 was used at the loan request process at the client side, at the reception of the contract message from the bank, this relation matches with putting the initiate parameter of the correlation set "`loanRequest`" at "no" and the one for the correlation set "`contract`" at "yes".

4. **invoke with reply:** In the BPEL process example at the bank side, we present two types of invoke activities:

   - the simple invoke in BPEL, is a one way request at the partner side, this activity can define an input variable without expecting a reply in an out variable. In our model, a simple invoke is presented by :



("a": correlation set `loanRequest`, "b": message content)

Figure 4.2: R, R1, R2 and R3 pre-orders

$$\texttt{l}!m(x)$$

- the invoke/reply activity in BPEL is presented as an invoke activity which defines an input and output variables. This invoke type waits for reply from the partner, the reply message will be assigned to the output variables.

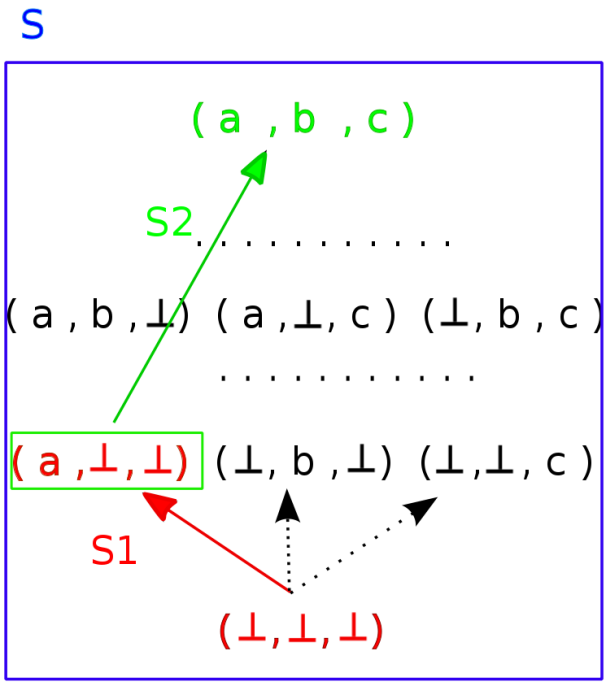
□

Example 4.2: (Part 2: Making a trade between two clients and the bank)

Consider a bank business process that does its work after two clients have reached an agreement on a particular trade. Both trading partners inform the bank who then initiates the settlement. The bank process waits for a message from both partners without knowing which one arrives first. In BPEL, this form of multiple event processing is presented by using the parameter "`join`".

In Listing 4.3 we present briefly the BPEL process structure.

This BPEL example presents a correlation set "`tradeID(clientAID, clientBID)`", which is used to create an instance of the BPEL process for others trades made between other couples of



("a": correlation set `loanRequest`, "b": correlation set `contract`, "c": message content)

Figure 4.3: S, S1 and S2 pre-orders

clients. The idea is that, if two client A and B are agreeing to a trade and they wish to send it to the bank, each client send a message by giving his id and the partner id. When the bank receives the first message, he will create a process with initiating a correlation set by combining clientAID and clientBID gotten from the incoming message. Next the bank will send the correlation set value to clients to use it in their future messages.

```
1  <process name="LoanProcess" ...>
2     <correlationSets>
3        <correlationSet name="tradeID"
4           properties="cor:clientAID cor:clientBID" />
5     </correlationSets>
6       ...
7     <flow>
8        <receive partnerLink="clientA"
9           operation="receiveClientAInformation"
10          inputVariable="clientInformation"
```

```
11          createInstance="yes"  >
12          ...
13          <correlations>
14             <correlation set="tradeID" initiate="join"/>
15          </correlations>
16       </receive>
17
18       <receive partnerLink="clientB"
19          operation="receiveClientBInformation"
20          inputVariable="clientInformation"
21          createInstance="yes" ... >
22          ...
23          <correlations>
24             <correlation set="tradeID" initiate="join"/>
25          </correlations>
26       </receive>
27          ...
28    </flow>
29 </process>
```

Listing 4.3: BPEL example for multiple event processing

The translation of the BPEL example 4.3 in our model must respect the following principles:

- Suppose that the two parallel receive activities are represented by two activities a_1 and a_2 in our model, such as, a_1 is designed to receive on the channel `"receiveClientAInformation"` and a_2 is designed to receive on the channel `"receiveClientBInformation"`. These two activities will not presented in parallel in our model, but we'll choose a randomly order between a_1 and a_2, and we will put them in sequence.

- If we choose to put a_1 before of a_2, the process will start only when he will receive a message on the channel `receiveClientAInformation`. The activity a_2 will be launched next only at the reception of a message on the channel `receiveClientBInformation` containing the same value of the correlation set received by the activity a_1.

- For all incoming messages that can't be received by the waiting activity or activities, they will be putted in the inbox "IN".

- Each time a process is started at the reception of a message, we must browse the "IN" inbox, to check if the process has received a message before which matches with the current process instance. For example, suppose that, the process receive the first message from the clientB:

$$clientInformation = pair(myID(12))(partnerID(14))$$

As a_1 is the first waiting activity, the incoming message will be putted in a waiting state in the inbox "IN". Suppose after a time t, the process will receive a message on the channel `receiveClientAInformation` the following message:

$$clientInformation = pair(myID(14))(partnerID(12)).$$

The process will create an instance with the correlation set "`tradeID(14,12)`". Next, the process will browse the inbox IN and he will take the previous stocked message received from the clientB.

A more detailed description of this example will be given in future works. □

Chapter 5

Outlook: service evolutions, aspects, and properties

The CESSA consortium investigates the preservation and enforcement of security properties of service-oriented architectures that are subject to evolution task expressed using aspects. The formal model introduced above provides basic machinery for the definition of collaborations (*i.e.,* choreographies) and orchestrations, but does not define aspects and collaboration-global properties. While these two issues are not a subject of this deliverable, we briefly present how aspects and global properties fit into the picture in the remainder of this section.

Concretely, we envision to define, analyze and enforce properties of service compositions in terms of protocols. Service evolution will be defined in terms of history-based explicitly-distributed aspects. In the following we informally describe some characteristics of their corresponding definitions as well as potential corresponding approaches on which our aspect and property extensions will be based on.

The corresponding formal models of aspect-based service evolutions and properties of service compositions will be delivered as part of the next phase of the project, the definition of a concrete language for the secure evolution of services using aspects.

5.1 Protocols for the definition of collaboration-global properties

We intend to use the formalism of session types [15, 7] as a basic representation of collaboration-global. As presented in the discussion of the state-of-the-art (Sec. 2.2.1), session types have four main characteristics (C1–4, page 9): the provide a global protocol definition that can be projected (and checked) on individual sites, support different communication and interaction schemes, and enable several important properties (such as compatibility) to be ensured through static analysis.

All of these properties are clearly relevant in the context of the CESSA project and can be used to meet the following goals:

- **Global properties, local enforcement.** The notion of a global property provides a useful

notion on which to build explicit definitions of global properties. The projection of session protocols onto individual sites constitutes an interesting mechanism that will be used to enforce service properties locally.

- **Flexible service composition.** Session types support synchronous and asynchronous communication as well as event-based styles of programming. These features come in handy in order to express the evolution using aspects that may add, express and modify interactions as well as events occurring as part of the horizontal and vertical compositions of service-based systems.

- **Analysis of composition properties.** We intend to define analyses of service composition properties, notably security properties of service compositions, on top of session types and the basic properties they define natively.

Technically, we will consider the integration of our service and aspect models with the most recent variants of session types that provide support for asynchronous session types and multiparty session types [17], as well as support for event-based programming [18]. This integration mainly impacts the collaboration part of our calculus but also requires modification of the process language, in particular because correlation is an essential feature in the definition of service compositions.

5.2 Aspects for service evolution

The CESSA project investigates the evolution of (security properties) of service-oriented servers that involve large-scale server infrastructures and applications executing on those server infrastructures but also on, *e.g.,* mobile devices. A mechanism for the definition of the corresponding evolution scenarios has to the following basic requirements of reconciling *flexibility and the tractability of property analysis and enforcement*: evolution scenarios frequently consist of multiple evolution tasks that may affect large parts of the underlying system or modify fundamental but small parts of the structure or behavior of a system; at the same time, the preservation of properties (of correctness, security, ...) of all of these modifications have to be statically or possibly dynamically checkable.

In the CESSA project, aspects are used as the basic concept for the definition of evolution scenarios in order to cope with crosscutting functionalities that are very common in service-oriented systems. Aspects typically provide a large degree of flexibility and often include mechanisms for the modification of large to very small parts of software systems. Some aspect systems for the manipulation of service-oriented systems provide such a degree of flexibility. However, no aspect model for service-oriented systems has been amenable to-date to property analysis and enforcement.

In order to support a flexible but property-proof aspect model for our service-oriented model, the aspect model is required to meet the following requirements:

- **History-based execution contexts.** Collaborations, *i.e.,* the highest-level horizontal service compositions, are defined in terms of, frequently complex, interactions involving sev-

eral partners. Properties over collaborations, in particular if expressed using protocols as outlined in Sec. 5.1, are also based on sequences of multiparty interactions. Finally, evolutions involving modifications on several layers of abstractions within the service software stack frequently do likewise.

Consequently, the definition of execution contexts that define where evolutions modify a system (the so-called pointcuts in an aspect-based system) should also be expressible in terms of multiparty event sequences.

- **Support for explicit distribution.** Collaborations as well as interactions involving different service and implementation levels may involve services and resources from different locations. In the collaboration calculi, for instance, locations are explicit (see Def. 3.1).

  The aspect model should therefore provide explicit means for the representation of distributed entities, both as part of the definition of execution contexts (the pointcut part of aspects) and the definition of modifications that implement evolution tasks (the advice part of aspects).

- **Flexible service composition.** Similar to the service model introduced before and the protocol-based properties outlined in Sec. 5.1, the aspect model must support flexible synchronization and communication relationships between distributed events.

There are aspect models that partially address these requirements: history-based pointcuts have been proposed in the context of functional programming languages [10] and several aspect languages over protocol-like structures have been proposed [9, 41, 31]. Aspects with explicit distribution have been proposed, in particular, on top of Java [32] and, recently, a first formal modal [37]. The AWED aspect model [30] supports, in addition, flexible communication and synchronization of distributed events. None of these approaches supports, however, service-specific concepts, in particular notions of distribution that have to take into account correlation and vertical service compositions. We will provide an extension of our service model (and protocol-based model for service properties) by a model of history-based explicitly-distributed aspects that partially specializes, partially extends these previous aspect models.

Chapter 6

Conclusion

In this deliverable we have provided a formal framework for the CESSA service model introduced in deliverable D1.1. This framework is structured in two layers: a collaboration layer and a process layer. We have instantiated this framework by defining a process model based on the π-calculus. We have shown how to apply the model to the formalization of a significant subset of BPEL, a standard language for service orchestration. Overall, our approach provides two main contributions: (i) the collaboration model provides a generalization of existing models of service choreography and (ii) our BPEL formalization includes a consistent set of features in a concise way.

Furthermore, we have presented first steps towards the extension of the service framework by formally-defined collaboration protocols and a formal model for evolution aspects. The resulting model for the secure evolution of services will be developed and applied to the service-oriented applications from partners SAP and IS2T in the next phase of the project.

Bibliography

[1] Faisal Abouzaid and John Mullins. A calculus for generation, verification and refinement of BPEL specifications. *Electr. Notes Theor. Comput. Sci*, 200(3):43–65, 2008.

[2] Faisal Abouzaid and John Mullins. Model-checking web services orchestrations using BP-calculus. *Electr. Notes Theor. Comput. Sci*, 255:3–21, 2009.

[3] Faisal Abouzaid and John Mullins. Formal specification of correlation in WS orchestrations using BP-calculus. *Electr. Notes Theor. Comput. Sci*, 260:3–24, 2010.

[4] Michele Boreale, Roberto Bruni, Luís Caires, Rocco De Nicola, Ivan Lanese, Michele Loreti, Francisco Martins, Ugo Montanari, António Ravara, Davide Sangiorgi, Vasco Thudichum Vasconcelos, and Gianluigi Zavattaro. SCC: A service centered calculus. In Mario Bravetti, Manuel Núñez, and Gianluigi Zavattaro, editors, *Web Services and Formal Methods, Third International Workshop, WS-FM, Proceedings*, volume 4184 of *Lecture Notes in Computer Science*, pages 38–57. Springer, 2006.

[5] Roberto Bruni. Calculi for service-oriented computing. In Marco Bernardo, Luca Padovani, and Gianluigi Zavattaro, editors, *Formal Methods for Web Services, 9th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2009, Bertinoro, Italy, June 1-6, 2009, Advanced Lectures*, volume 5569 of *Lecture Notes in Computer Science*, pages 1–41. Springer, 2009.

[6] Javier Cámara, Carlos Canal, Javier Cubo, and Antonio Vallecillo. Formalizing WSBPEL business processes using process algebra. *Electronic Notes in Theoretical Computer Science*, 154(1):159–173, 2006.

[7] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centred programming for web services. In Rocco De Nicola, editor, *Programming Languages and Systems, 16th European Symposium on Programming, ESOP 2007, Held as Part of the Joint European Conferences on Theory and Practics of Software, ETAPS 2007, Braga, Portugal, March 24 - April 1, 2007, Proceedings*, volume 4421 of *Lecture Notes in Computer Science*, pages 2–17. Springer, 2007.

[8] Oasis Consortium. Universal Description, Discovery, and Integration specification. `http://uddi.org/pubs/uddi-v3.0.2-20041019.pdf`.

[9] Rémi Douence, Pascal Fradet, and Mario Südholt. Trace-based aspects. In Mehmet Akşit, Siobhán Clarke, Tzilla Elrad, and Robert E. Filman, editors, *Aspect-Oriented Software Development*, pages 201–218. Addison-Wesley Professional, September 2004.

[10] Rémi Douence, Olivier Motelet, and Mario Südholt. A formal definition of crosscuts. In *Proceedings of the 3rd International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, volume 2192 of *LNCS*, pages 170–186. Springer Verlag, September.

[11] Rémi Douence, Hervé Grall, Ismael Mejia, et al. Survey and requirements analysis. Deliverable D1.1, The CESSA consortium, July 2010.

[12] Schahram Dustdar and Mike P. Papazoglou. Services and service composition - an introduction (services und service komposition - eine einführung). *it - Information Technology*, 50(2):86–92, 2008.

[13] Fabrício Fernandes, Robin Passama, and Jean-Claude Royer. Event strictness for components with complex bindings. In Kiran Deshpande, Pankaj Jalote, and Sriram K. Rajamani, editors, *ISEC'09: Proceedings of the 2nd conference on India Software Engineering Conference*, pages 47–56. ACM, February 2009.

[14] Andrea Ferrara. Web services: a process algebra approach. In Marco Aiello, Mikio Aoyama, Francisco Curbera, and Mike P. Papazoglou, editors, *Service-Oriented Computing - ICSOC 2004, Second International Conference, New York, NY, USA, November 15-19, 2004, Proceedings*, pages 242–251. ACM, 2004.

[15] Simon J. Gay, Vasco Thudichum Vasconcelos, António Ravara, Nils Gesbert, and Alexandre Z. Caldeira. Modular session types for distributed object-oriented programming. In Manuel V. Hermenegildo and Jens Palsberg, editors, *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010*, pages 299–312. ACM, 2010.

[16] G. Serme A. Santana De Oliveira S. Idrees Y. Roudier G.Harel. Security analysis for web services, deliverable 3.1. ANR project no. 09-SEGI-002-01, 2010.

[17] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In George C. Necula and Philip Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 273–284. ACM, 2008.

[18] Raymond Hu, Dimitrios Kouzapas, Olivier Pernet, Nobuko Yoshida, and Kohei Honda. Type-safe eventful sessions in java. In Theo D'Hondt, editor, *ECOOP 2010 - Object-Oriented Programming, 24th European Conference, Maribor, Slovenia, June 21-25, 2010. Proceedings*, volume 6183 of *Lecture Notes in Computer Science*, pages 329–353. Springer, 2010.

[19] David Kitchin, Adrian Quark, William R. Cook, and Jayadev Misra. The Orc programming language. In David Lee, Antónia Lopes, and Arnd Poetzsch-Heffter, editors, *Proceedings of FMOODS/FORTE 2009*, volume 5522 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 2009.

[20] Mariya Koshkina and Franck van Breugel. Modelling and verifying web service orchestration by means of the concurrency workbench. *SIGSOFT Softw. Eng. Notes*, 29(5):1–10, 2004.

[21] A. Lapadula, R. Pugliese, and F. Tiezzi. A Calculus for Orchestration of Web Services. In *Proc. of 16th European Symposium on Programming (ESOP'07)*, volume 4421 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2007.

[22] Alessandro Lapadula, Rosario Pugliese, and Francesco Tiezzi. A WSDL-based type system for WS-BPEL. In Paolo Ciancarini and Herbert Wiklicky, editors, *Coordination Models and Languages, 8th International Conference, COORDINATION 2006, Bologna, Italy, June 14-16, 2006, Proceedings*, volume 4038 of *Lecture Notes in Computer Science*, pages 145–163. Springer, 2006.

[23] Alessandro Lapadula, Rosario Pugliese, and Francesco Tiezzi. Cows: a timed service-oriented calculus. In *ICTAC'07: Proceedings of the 4th international conference on Theoretical aspects of computing*, pages 275–290, Berlin, Heidelberg, 2007. Springer-Verlag.

[24] Alessandro Lapadula, Rosario Pugliese, and Francesco Tiezzi. A formal account of ws-bpel. In *COORDINATION'08: Proceedings of the 10th international conference on Coordination models and languages*, pages 199–215, Berlin, Heidelberg, 2008. Springer-Verlag.

[25] L. Logrippo, M. Faci, and M. Haj-Hussein. An Introduction to LOTOS: Learning by Examples. *Computer Networks and ISDN Systems*, 23:325–342, 1992.

[26] Niels Lohmann, Eric Verbeek, and Remco M. Dijkman. Petri net transformations for business processes - A survey. *T. Petri Nets and Other Models of Concurrency*, 5460:46–63, 2009.

[27] Niels Lohmann, Eric Verbeek, Chun Ouyang, and Christian Stahl. Comparing and evaluating petri net semantics for BPEL. *Int. J. of Business Process Integration and Management*, 4:60–73, July 12 2009.

[28] Roberto Lucchi and Manuel Mazzara. A pi-calculus based semantics for WS-BPEL. *J. Log. Algebr. Program*, 70(1):96–118, 2007.

[29] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes I. *Journal of Information and Computation, Academic Press*, 100(1):1–40, Sept. 1992.

[30] Luis Daniel Benavides Navarro, Mario Südholt, Wim Vanderperren, Bruno De Fraine, and Davy Suvée. Explicitly distributed aop using awed. In Robert E. Filman, editor, *AOSD*, pages 51–62. ACM, 2006.

[31] Dong Ha Nguyen and Mario Südholt. VPA-based aspects: better support for AOP over protocols. In *4th IEEE International Conference on Software Engineering and Formal Methods (SEFM'06)*. IEEE Press, September 2006.

[32] Muga Nishizawa, Shigeru Chiba, and Michiaki Tatsubori. Remote pointcut - a language construct for distributed aop. In Karl Lieberherr, editor, *Proceedings of the 3rd ACM Int. Conf. on Aspect-Oriented Software Development (AOSD), Lancaster*. ACM Press, 2004.

[33] Oasis Consortium. Web Services Business Process Execution Language Version 2.0. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf`, 11 April, 2007.

[34] Pascal Poizat and Jean-Claude Royer. A Formal Architectural Description Language based on Symbolic Transition Systems and Modal Logic. *Journal of Universal Computer Science*, 12(12):1741–1782, 2006.

[35] Alberto Portilla, Genoveva Vargas-Solar, José-Luis Zechinelli-Martini, Christine Collet, and Luciano García-Bañuelos. A survey for analyzing transactional behavior in service based applications. In *Seventh Mexican International Conference on Computer Science ('ENC)*, pages 116–126. IEEE Computer Society, 2006.

[36] Sidney Rosario, David Kitchin, Albert Benveniste, William Cook, Stefan Haar, and Claude Jard. Event structure semantics of Orc. In Marlon Dumas and Reiko Heckel, editors, *Web Services and Formal Methods*, volume 4937 of *Lecture Notes in Computer Science*, pages 154–168, 2008.

[37] Nicolas Tabareau. A theory of distributed aspects. In *Proceedings of the 9th International Conference on Aspect-Oriented Software Development*, AOSD '10, pages 133–144, New York, NY, USA, 2010. ACM.

[38] Franck van Breugel and Maria Koshkina. Models and verification of bpel. 2006.

[39] Hugo Torres Vieira, Luís Caires, and João Costa Seco. The conversation calculus: A model of service-oriented computation. In Sophia Drossopoulou, editor, *Programming Languages and Systems, 17th European Symposium on Programming, ESOP 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, Proceedings*, volume 4960 of *Lecture Notes in Computer Science*, pages 269–283. Springer, 2008.

[40] Mirko Viroli. A core calculus for correlation in orchestration languages. *J. Log. Algebr. Program*, 70(1):74–95, 2007.

[41] Robert J. Walker and Kevin Viggers. Implementing protocols via declarative event patterns. In *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE-12)*, pages 159 – 169. ACM Press, 2004.

[42] World Wide Web Consortium. *Web Services Choreography Description Language Version 1.0*, 2005. `http://www.w3.org/TR/ws-cdl-10/`.

[43] Hongli Yang, Xiangpeng Zhao, Zongyan Qiu, Geguang Pu, and Shuling Wang. A formal model forweb service choreography description language (WS-CDL). In *International Conference on Web Services (ICWS'06)*, pages 893–894. IEEE Computer Society, 2006.

[44] Yongxin Zhao, Zheng Wang, Geguang Pu, and Huibiao Zhu. A formal model for service choreography with exception handling and finalization. In *Fourth International Symposium on Theoretical Aspects of Software Engineering (TASE'10)*. IEEE Computer Society, 2010.