Formal Enforcement of Security Policies on Choreographed Services

Mahjoub Langar and Karim Dahmani

LIP2 Research Laboratory, Faculté des Sciences de Tunis, Tunisia

Abstract. Web services are software systems that support distributed applications composed of independent processes which communicate by message passing. To realize the full potential of web services, we need to compose existent web services in order to get more functionalities. However the composition of web services should be secure. In this paper we propose an automatic formal approach to monitor the execution of a choreography of web services and we prove its correctness. We introduce the syntax and semantic rules of a new operator which takes as input a choreography and a security policy and produces as output a secure version of this choreography which behaves like the original one and does not violate the security policy.

Keywords: Choreographed services, web service security, formal verification, runtime verification

1 Introduction

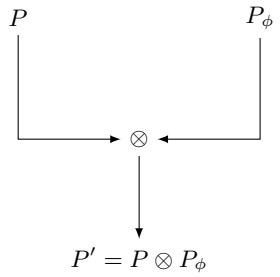
With the explosive growth of the Internet, intranet and electronic commerce, the concept of web services has emerged in the world to be exploited by most cross-organizational boundaries. In fact, web services provide a suitable technical foundation for making business processes accessible within and across enterprises. Moreover web services are software systems that support distributed applications composed of independent processes which communicate by message passing. But for an appropriate exploitation of web services, we need to compose them. Indeed, individual web services often offer limited capabilities. In some situations, a client's request cannot be satisfied by a single web service. However, when composed with other web services, they can satisfy the client demand. To realize the full potential of web services, we need to compose existent web services in order to get more functionalities. Web service composition rules deal with how different services are composed into a coherent global service. In particular, they specify the order in which services are invoked, and the conditions under which a certain service may or may not be invoked. Among the approaches investigated in service composition, we distinguish orchestration and choreography. The orchestration composes available services and adds a central coordinator (the orchestrater) which is responsible for invoking and composing the single sub-activities. However the second one, referred to as web

service choreography, does not assume the exploitation of a central coordinator but it defines complex tasks via the definition of the conversation that should be undertaken by each participant. While several proposals exist for orchestration languages (e.g. Business Process Execution Language (BPEL)[1], Web Services Flow Language (WSFL)[2]), choreography languages are still in a preliminary stage of definition. A proposal, Web Service Choreography Description Language (WS-CDL)[3], was issued by the World Wide Web Consortium (W3C) in December 2004.

The need of secure web service composition has led to a great interest from researchers of the last decade. However most of the works focus on how to guarantee desired properties such as correctness, deadlock avoidance, etc. In our work, in addition to these properties, we introduce an automatic approach to monitor a choreography by specifying and enforcing security policies on web services and we prove its correctness using formal methods. Some works such as [4,5] have enforced security policies on concurrent systems using process algebra. Formal methods are mathematical techniques that are well-suited to address the aforementioned issues. A variety of proposals to formally describe, compose and verify web service compositions using the formal methods exists. In [6], a Petri-net based design and verification framework for web service composition is proposed. In [7], the authors introduce a Petri-net based algebra to compose web services based on control flows. In [8], authors use Milner's process algebra CCS to specify and compose web services as processes, and the Concurrency Workbench to validate properties such as correct web service composition. In [9], the authors provide an encoding of BPEL processes into web service timed state transition systems, a formalism that is closely related to timed automata and discuss a framework in which timed assumptions can be model checked.

In this paper, we propose an enforcement of security policies over web services using formal methods. We present an automated approach for monitoring the behavior of a service in a choreography. More precisely, we define an operator $\otimes$ that takes as input a process P and a security policy P_ϕ and generates a new process $P' = P \otimes P_\phi$ which respects the following conditions :

- $P' \mid\sim P_\phi$, i.e. P' satisfies the security policy P_ϕ.
- $P' \sqsubseteq P$, i.e. behaviors of P' are also behaviors of P.
- $\forall Q \ : \ ((Q \mid\sim P_\phi)$ and $(Q \sqsubseteq P)) \Longrightarrow Q \sqsubseteq P'$, i.e. all good behaviors of P are also behaviors of P'.



The rest of the paper is structured as follows. In section 2, we briefly describe the global calculus. Section 3 presents the formalism used to specify security policies. Section 4 illustrates the formalism used to specify choreographed services. In section 5, we present our security framework for monitoring a choreography. In section 6, we discuss some related works while conclusions are in section 7.

2 Global Calculus

The global calculus [10] is distilled from WS-CDL. It describes the global flow of interactions between the participants. The syntax of the global calculus is given by the standard BNF.

2.1 Syntax of the Global Calculus

$$
\begin{array}{rll}
I ::= & A \to B : ch(\nu \tilde{s}).I & (Init) \\
| & A \to B : s\langle op, e, y\rangle.I & (Comm) \\
| & x@A := e.I & (Assign) \\
| & if\ e@A\ then\ I_1\ else\ I_2 & (IfThenElse) \\
| & I_1 + I_2 & (Sum) \\
| & I_1 | I_2 & (Par) \\
| & (\nu s)I & (New) \\
| & X^A & (recVar) \\
| & rec\ X^A.I & (Rec) \\
| & 0 & (Inaction)
\end{array}
$$

The first rule $(Init)$ says that the participant A invokes a service ch located at B and initiates new freshly generated session channels $\tilde{s}$. The second rule $(Comm)$ expresses the sending action by A whose message consists of a selected operator op and an expression e to the participant B which will store the value of e at the variable y. $(Assign)$ is a local construct which updates the variable x located at A with e. (Par) and (Sum) are respectively the parallel composition and the non-deterministic choice of interactions. $(IfThenElse)$ is the standard conditional operation, while $(recVar)$ and (Rec) are used to express recursive behavior of interactions. 0 is the inactive interaction.

2.2 Semantics of the Global Calculus

The formal semantics of the global calculus is defined in terms of *configurations* (σ, I). The notation $(\sigma, I) \rightarrow (\sigma', I')$ says that the global description I at a state σ (which is the collection of all local states of the participants) will reduce to I' at a new state σ'. Samples of reduction rules are presented here while the overall operational semantic is detailed in [10].

$$(Init)\frac{-}{(\sigma, A \rightarrow B : ch(\nu \tilde{s}).I) \rightarrow (\sigma, (\nu \tilde{s})I)}$$

$$(Comm)\frac{\sigma \vdash e@A \Downarrow v}{(\sigma, A \rightarrow B : s\langle op, e, x\rangle.I) \rightarrow (\sigma[x@B \mapsto v], I)}$$

$$(Assign)\frac{\sigma \vdash e@A \Downarrow v}{(\sigma, x@A := e.I) \rightarrow (\sigma[x@A \mapsto v], I)} \quad (Par)\frac{(\sigma, I_1) \rightarrow (\sigma', I_1')}{(\sigma, I_1|I_2) \rightarrow (\sigma', I_1'|I_2)}$$

$$(Sum)\frac{(\sigma, I_1) \rightarrow (\sigma', I_1')}{(\sigma, I_1 + I_2) \rightarrow (\sigma', I_1')} \quad (Rec)\frac{(\sigma, I[rec\ X^A.I/X^A]) \rightarrow (\sigma', I')}{(\sigma, rec\ X^A.I) \rightarrow (\sigma', I')}$$

The first rule is for initiation : A invokes a service located at B and initiates session channels $\tilde{s}$ which will be shared locally by A and B. The second rule is for communication. The notation $\sigma \vdash e \Downarrow v$ is an evaluation judgment : σ evaluates the expression e to the value v. $\sigma[x@B \mapsto v]$ is the resulting state of updating the local variable x at B by v. The evaluation judgment is also used in the assignment rule. The fourth rule is for parallelism. The (Sum) rule describes the behavior of a non-deterministic choice. The (Rec) rule says that if the unfolding of $rec\ X^A.I$ under σ reduces to I' then $rec\ X^A.I$ under σ will reach (σ', I').

Example 1. This example shows a communication between a buyer and a seller. These participants share new freshly generated session channels $B2Sch$ and $S2Bch$. Through $S2Bch$, the seller offers a quote to the buyer. Through $B2Sch$, the buyer selects one of the two options offered by the seller, QuoteAccept and QuoteReject. If the first option is selected, the buyer sends the quote "100" which will be stored in x by Seller and continues with the interaction I_1. In the other case, the seller sends the abort number stored in the variable $x_{AbortNo}$ which will be stored in y by the Seller and terminates.

Buyer $\rightarrow$ Seller : $ch(\nu B2Sch, S2Bch)$.
Seller $\rightarrow$ Buyer : $S2Bch\langle quote, 100, y\rangle$.
if $y@Buyer \leq 1000$ then
  { Buyer $\rightarrow$ Seller : $B2Sch\langle quoteAccept, 100, x\rangle.I_1$ }
else
  { Buyer $\rightarrow$ Seller : $B2Sch\langle quoteReject, x_{abortNo}, x\rangle.0$ }

3 Security Policy Specification

The language that we will use for the specification of security policies is EPC^{ϕ} which is a subset of EPC (End-point calculus). The End-Point Calculus precisely identifies a local behavior of each participant in a web service. End-Point

Calculus is inspired from the π-calculus. It is an applied variant of π-calculus augmented with the notion of participants and their local states. In [10], authors have established a projection from the global calculus into the end-point calculus. So descriptions of participants behaviors in end-point calculus are not extracted directly from the choreography, but projected from the global calculus. Details of this theory of end-point projection are presented in [10]. Since in the proposed enforcement approach we transform the security policy to a monitor, we reach immediately this goal by choosing EPC^{ϕ} since it is a subset of EPC.

3.1 Syntax of EPC^{ϕ}

We show the syntax of EPC^{ϕ}, where s denote session channels, e an expression which can be an atomic value (such as a natural number or a boolean value) or a function (such as arithmetic functions and boolean operators) or a variable. $op_1, op_2, ...$ range over operations. x is a variable. The first construct is a receiving action, it is the invocation of one of the operators op_i and reception of an expression which will be evaluated and stored in x_i. The second construct is a sending action, it is the invocation of operator op with expression e. Furthermore, the operator "$\oplus$" represents the alternative composition. Finally, recursion is used for representing unbounded repetition. For representing recursive behaviors, we use term variables $X, Y, \ldots$ The operator used for recursion is rec $X.P$. Each occurrence of X in P denotes a recurring point. rec $X.P$ behaves as P until an occurrence of X is found in the execution of P, then it will return to rec $X.P$.

$$s \rhd \sum_i op_i(x_i).P_i, \quad \overline{s} \lhd op\langle e\rangle.P, \quad P_1 \oplus P_2, \quad X, \quad \text{rec } X.P, \quad 0$$

3.2 Semantics of EPC^{ϕ}

Note that processes do not evolve alone. A process is located in a participant which synchronize with another one to evolve. A participant A with its behavior P at a local state σ is called a network and denoted by $A[P]_{\sigma}$. Syntax of networks is given by the following grammar :

$$\begin{array}{rrl} N ::= & A[P]_{\sigma} & \text{(Participant)} \\ | & N|M & \text{(Parallel-NW)} \\ | & (\nu s)N & \text{(Res-NW)} \\ | & \epsilon & \text{(Inaction-NW)} \end{array}$$

where $A[P]_{\sigma}$ is a participant with its behavior as shown in the first construct. Two communicating participants are represented by two networks combined by parallel composition. When two participants initiate a session channel for communication, this session channel must be restricted to these two participants, this is given by (Res-NW). ϵ denote the lack of networks. Reduction rules of networks are given by :

$$\frac{M \equiv M' \quad M' \rightarrow N' \quad N' \equiv N}{M \rightarrow N} \text{Struct-NW}$$

$$\frac{M \to M'}{M|N \to M'|N}\text{Par-NW} \quad \frac{M \to M'}{(\nu s)M \to (\nu s)M'}\text{Res-NW}$$

Semantics of EPC^{ϕ} is then given by :

$$\frac{\sigma_2 \vdash e \Downarrow v}{A[\overline{s} \lhd op_j\langle e\rangle.P]_{\sigma_1}|B[s \rhd \sum_i op_i(x_i).Q_i]_{\sigma_2} \to A[P]_{\sigma_1}|B[Q_j]_{\sigma_2[x_j \mapsto v]}}$$

$$\frac{A[P_1]_{\sigma} \to A[P_1']_{\sigma'}}{A[P_1 \oplus P_2]_{\sigma} \to A[P_1']_{\sigma'}} \quad \frac{A[P[\text{rec } X.P/X]]_{\sigma} \to A[P']_{\sigma'}}{A[\text{rec } X.P]_{\sigma} \to A[P']_{\sigma'}}$$

Security policies are usually specified in logic-based languages. Such languages employ mainly three operators to compose properties : *and, or* and *not*. The *or* operator is given here by $\oplus$. The *not* operator is defined here as follows : assume we have two participants A and B communicating and we want to apply the property "A should not send op_1 to participant B", then we write it using EPC^{ϕ} as follows :

$$P = \text{rec } X.(\overline{s_B} \lhd \bigoplus_{i \neq 1} op_i\langle e_i\rangle.X \oplus s_B \rhd \sum_i op_i(x_i).X)$$

Example 2. Assume we have a client wanting to check its account details within a bank. The bank should not send him back his account details if he is not yet authenticated. A client is said to be authenticated if he has received from the bank an acceptance for his authentication's attempt using the operation *accept*. The bank answers the client for his account request through the operation *resAccount*. The security property will then be written as follows :

$$P_{\phi} = \text{rec } X.(s \rhd \sum_i op_i(x_i).X \quad \oplus$$

$$\overline{s} \lhd \bigoplus_{op_i \notin \{accept, resAccount\}} op_i\langle e\rangle.X \quad \oplus$$

$$\overline{s} \lhd accept.\text{rec } Y.(\overline{s} \lhd \bigoplus_i op_i\langle e_i\rangle.Y \oplus s \rhd \sum_i op_i(x_i).Y))$$

The security property is expressed using the recursion operator. In the recursion block, we have 3 behaviors combined with the internal choice $\oplus$ which we denote by P_{ϕ_1}, P_{ϕ_2} and P_{ϕ_3} where

$$P_{\phi_1} = s \rhd \sum_i op_i(x_i).X$$

$$P_{\phi_2} = \overline{s} \lhd \bigoplus_{op_i \notin \{accept, resAccount\}} op_i\langle e\rangle.X$$

$$P_{\phi_3} = \overline{s} \lhd accept.\text{rec } Y.(\overline{s} \lhd \bigoplus_i op_i\langle e_i\rangle.Y \oplus s \rhd \sum_i op_i(x_i).Y)$$

The block P_{ϕ_1} expresses the fact that the property allows all receiving actions. The block P_{ϕ_2} inhibits sending *accept* or *resAccount* to the client. The block P_{ϕ_3} intercepts the *accept* sending action and then no more restrictions are imposed since the authentication of the client is accepted.

So, an intruder who wants to check an account's details without authentication cannot achieve its goal since the *resAccount* is only permitted in the block following the *accept* sending action.

4 Choreography Specification

The technique used in this paper for describing the composition of web services is the choreography. The WS-CDL is based on two engineering principles :

Service Channel Principle corresponds to the repeated availability of service channels.

Session Principle is a basic principle in many communication-centred programs which says a sequence of conversations belonging to a protocol should not be confused with other concurrent runs of this or other protocols by the participants.

As a specification of the choreography description language WS-CDL, we introduce a modified version of the end-point calculus which is a formalism presented in [10]. The end-point calculus describes the behavior of a participant in a choreography from its end-point view.

4.1 Secured End-Point Calculus

The secured end-point calculus EPC_S^ϕ is a variant of EPC (End-Point Calculus)[10]. EPC_S^ϕ has the particularity of explicitly handling the monitoring concept through its operator ∂_{P_ϕ}. For instance, the process $\partial_{P_\phi}(P)$ can execute only actions that could be executed by both the controller P_ϕ and the process P. We describe hereafter the formal syntax and dynamic semantics of the secured end-point calculus.

Syntax of EPC_S^ϕ Since EPC^ϕ is a subset of EPC and EPC_S^ϕ is an extension of EPC, a part of the syntax of EPC_S^ϕ is presented in the last section. So in this section we will present the overall syntax of EPC_S^ϕ but we will describe only constructs that have not been described before.

$$
\begin{array}{rl}
P ::= & !\underline{ch}(\tilde{s}).P \\
| & \overline{ch}(\nu\tilde{s}).Q \\
| & s \rhd \sum_i op_i(x_i).P_i \\
| & \overline{s} \lhd op\langle e\rangle.Q \\
| & x := e.P \\
| & \text{if } e \text{ then } P \text{ else } Q \\
| & P \oplus Q \\
| & P \mid Q \\
| & (\nu s)P \\
| & X \\
| & \text{rec } X.P \\
| & 0 \\
\hline
| & \partial_{P_\phi}(P)
\end{array}
$$

The two first constructs represent session initiation. $!ch(\tilde{s}).P$ is used for input and $\overline{ch}(\nu\tilde{s}).Q$ for output. $!ch(\tilde{s})$ says that the service channel ch, which is available to public, is ready to receive an unbounded number of invocations, offering a communication via its freshly generated session channels $s \in \tilde{s}$. $\overline{ch}(\nu\tilde{s})$ is an invocation of a service located at the service channel ch and an initiation of a communication session which will occur through session channels $\tilde{s}$. After a session has been initiated between two participants and freshly generated session channels have been shared between them, they can communicate using the communication constructs. The assignment operator is $x := e.P$. if e then P else Q is a choice based on the evaluation of the expression e. The parallel composition is given by $P|Q$. The restriction construct $(\nu s)P$ indicates that the session channel s is local to P. 0 denotes the lack of actions. Finally, $\partial_{P_\phi}(P)$ is the enforcement operator which controls the execution of P by allowing it to evolve if P_ϕ is satisfied.

Semantics of EPC_S^ϕ

Initiation/Communication simulation Let P be a process stipulating the local behavior of a participant A in a web service. We observe in the end-point calculus reduction rules that processes always evolve without an external factor unless in initiation and communication. In fact, a participant initiating a communication or communicating with an another participant needs to synchronize with him to realize the interaction . Therefore, for enforcing security policies on a participant's behavior, we need to have a totally local description of its interactions. So we need to define a simulation relation for the initiation and the communication of processes so that we can simulate the evolution of a process locally, i.e. without synchronizing with an another participant . We will define the normal form of a process then introduce the simulation relation.

Definition 1 (Normal form of a process). *Every process representing the local behavior of a participant in a web service can be written as an internal sum of processes, which we call the normal form of a process :*

$$\forall P \in \mathcal{P},\ P = \bigoplus_i a_i P_i$$

where $\mathcal{P}$ denotes the set of processes, a_i range over atomic actions and P_i range over processes in $\mathcal{P}$.

Definition 2 (Simulation Relation). *We define a simulation relation over networks, denoted by $A[P]_\sigma \overset{a}{\rightsquigarrow} A[P']_{\sigma'}$, which says that process P in A at the state σ is able to execute the action a and reduce to P' with a new local state σ'. The simulation relation is defined following this rule*

$$\frac{P = \bigoplus_i a_i P_i \quad \exists i \in \{1, \ldots, n\} : a = a_i}{A[P]_\sigma \overset{a}{\rightsquigarrow} A[P_i]_\sigma}$$

This rule says that when P is written in its normal form and one of the constituting processes is able to do an action a then $A[P]_\sigma$ is also able to do it.

Semantics of EPC_S^ϕ :

$$\text{Init}\quad \frac{-}{A[!ch(\tilde{s}).P \mid P']_{\sigma_A} \mid B[\overline{ch}(\nu\tilde{s}).Q \mid Q']_{\sigma_B} \to (\nu\tilde{s})(A[!ch(\tilde{s}).P|P|P']_{\sigma_A} \mid B[Q|Q']_{\sigma_B})}$$

$$\frac{\sigma_A \vdash e \Downarrow v}{A[x := e.P|P']_{\sigma_A} \to A[P|P']_{\sigma_A[x\mapsto v]}}\text{Assign}$$

$$\frac{\sigma_A \vdash e \Downarrow tt}{A[\text{if } e \text{ then } P_1 \text{ else } P_2|P']_{\sigma_A} \to A[P_1|P']_{\sigma_A}}\text{IfTrue}$$

$$\frac{\sigma_A \vdash e \Downarrow ff}{A[\text{if } e \text{ then } P_1 \text{ else } P_2|P']_{\sigma_A} \to A[P_2|P']_{\sigma_A}}\text{IfFalse}$$

$$\frac{A[P_1|R]_{\sigma_A} \to A[P_1'|R]_{\sigma_A'}}{A[P_1|P_2|R]_{\sigma_A} \to A[P_1'|P_2|R]_{\sigma_A'}}\text{Par} \qquad \frac{A[P]_{\sigma_A} \to A[P']_{\sigma_A'}}{A[(\nu s)P]_{\sigma_A} \to A[(\nu s)P']_{\sigma_A'}}\text{Res}$$

$$\text{Init-In-Sec}\quad \frac{-}{A[\partial_{P_\phi}(!ch(\tilde{s}).P)]_{\sigma_A}|B[\overline{ch}(\nu\tilde{s}).Q|R]_{\sigma_B} \to (\nu\tilde{s})(A[\partial_{P_\phi}(P)|\partial_{P_\phi}(!ch(\tilde{s}).P)]_{\sigma_A}|B[Q|R])}$$

$$\text{Init-Out-Sec}\quad \frac{-}{A[\partial_{P_\phi}(\overline{ch}(\nu\tilde{s}).P)]_{\sigma_A}|B[!ch(\tilde{s}).Q|R]_{\sigma_B} \to (\nu\tilde{s})(A[\partial_{P_\phi}(P)]_{\sigma_A}|B[!ch(\tilde{s}).Q|Q|R]_{\sigma_B})}$$

$$\frac{A[P]_{\sigma_A} \overset{s\triangleright op(x)}{\rightsquigarrow} A[P']_{\sigma_A} \quad A[P_\phi]_{\sigma_A} \overset{s\triangleright op(x)}{\rightsquigarrow} A[P'_\phi]_{\sigma_A} \quad \sigma_A \vdash e \Downarrow v}{A[\partial_{P_\phi}(P)]_{\sigma_A}|B[\overline{s} \lhd op\langle e\rangle.Q|R]_{\sigma_B} \to A[\partial_{P'_\phi}(P')]_{\sigma_A[x\mapsto v]}|B[Q|R]_{\sigma_B}}\text{Comm-In-Sec}$$

$$\text{Comm-Out-Sec}\quad \frac{A[P]_{\sigma_A} \overset{\overline{s}\lhd op_j\langle e\rangle}{\rightsquigarrow} A[P']_{\sigma_A} \quad A[P_\phi]_{\sigma_A} \overset{\overline{s}\lhd op_j\langle e\rangle}{\rightsquigarrow} A[P'_\phi]_{\sigma_A} \quad \sigma_B \vdash e \Downarrow v}{A[\partial_{P_\phi}(P)]_{\sigma_A}|B[s \rhd \sum_i op_i(x_i).Q_i|R]_{\sigma_B} \to A[\partial_{P'_\phi}(P')]_{\sigma_A}|B[Q_j|R]_{\sigma_B[x_j\mapsto v]}}$$

$$\frac{\sigma_A \vdash e \Downarrow v}{A[\partial_{P_\phi}(x := e.P)]_{\sigma_A} \to A[\partial_{P_\phi}(P)]_{\sigma_A[x\mapsto v]}}\text{Assign-Sec}$$

$$\frac{A[\partial_{P_\phi}(P_1)]_\sigma \to A[\partial_{P'_\phi}(P'_1)]_{\sigma'}}{A[\partial_{P_\phi}(P_1|P_2)]_\sigma \to A[\partial_{P'_\phi}(P'_1|P_2)]_{\sigma'}}\text{Par-Sec}$$

$$\frac{\sigma \vdash e \Downarrow tt}{A[\partial_{P_\phi}(P|if\ e\ then\ P_1\ else\ P_2)]_\sigma \to A[\partial_{P_\phi}(P|P_1)]_\sigma}\text{IfTrue-Sec}$$

$$\frac{\sigma \vdash e \Downarrow ff}{A[\partial_{P_\phi}(P|if\ e\ then\ P_1\ else\ P_2)]_\sigma \to A[\partial_{P_\phi}(P|P_2)]_\sigma}\text{IfFalse-Sec}$$

$$\frac{A[\partial_{P_\phi}(P)]_\sigma \rightarrow A[\partial_{P'_\phi}(P')]_{\sigma'}}{A[(\nu\tilde{s})(\partial_{P_\phi}(P))]_\sigma \rightarrow A[(\nu\tilde{s})(\partial_{P'_\phi}(P')]_{\sigma'})}\text{Res-Sec}$$

The Init-rule shows how two participants initiate a session by sharing new freshly generated session channels $\tilde{s}$. These session channels are restricted to participants A and B by the binding operator (ν). Assignment is a local construct. Assign-rule evaluates an expression e and assigns the result of this evaluation to the variable x in A, then A behaves as P. The Res-rule restricts the use of session channels $\tilde{s}$ to the process P in A. Init-In-Sec and Init-Out-Sec are the rules for communication initiation. We do not control session initiations but we control communication messages between the participants. Communication rules say if P_ϕ and P are able to send or receive through the same session channel the same operation and become respectively P'_ϕ and P' then $\partial_{P_\phi}(P)$ do this action and becomes $\partial_{P'_\phi}(P')$. Secured assignment rule says that assignment is not considered by enforcement. The secured parallel composition rule says that the security operator applied to $P_1|P_2$ can evolve into $(\partial_{P'_\phi}(P'_1|P_2))$ if P_1 can evolve simultaneously with the policy security P_ϕ into respectively P'_1 and P'_ϕ. For the restriction rule, binding session channels does not affect the enforcement. Finally, the conditional rules say that enforcement is not affected by conditionals.

5 Choreography Monitoring

The goal of this research is to enforce security policies over a choreography. Some of the important features of the enforcement operator is that it allows us to enforce only concerned participants by the security policies. In this in-lined monitoring framework, we do not modify the original behaviors of participants in a choreography. But if the security policy is not verified the evolution of the choreography will stop. In this section, we prove the correctness of our theory by defining first some notions such as the partial order over processes and satisfaction notions.

Definition 3 (Partial Order over Processes). *Let $A[P_1]_\sigma$, $A[P_2]_\sigma$ be two networks. We say $A[P_1]_\sigma \sqsubseteq A[P_2]_\sigma$ if the following condition hold :*

$$A[P_1]_\sigma \overset{a}{\rightsquigarrow} A[P'_1]_\sigma \implies A[P_2]_\sigma \overset{a}{\rightsquigarrow} A[P'_2]_\sigma \text{ and } A[P'_1]_\sigma \sqsubseteq A[P'_2]_\sigma.$$

Definition 4 (Satisfaction Notions). *Let P_ϕ be a security policy and ξ a trace. Symbols $\vDash$ and $|\sim$ are defined as follows :*

- *We say that ξ satisfies P_ϕ, denoted by $\xi \vDash P_\phi$, if $\xi \in [\![P_\phi]\!]$ where $[\![P_\phi]\!]$ denotes the set of traces of P_ϕ.*
- *We say that ξ could satisfy P_ϕ, denoted by $\xi \mid\sim P_\phi$, if it exists a trace ξ' such that $\xi.\xi' \vDash P_\phi$.*

Theorem 1. *Let P be a process and P_ϕ a policy security. The following properties hold :*

1. $\partial_{P_\phi}(P) \mid\sim P_\phi$,
2. $A[\partial_{P_\phi}(P)]_\sigma \sqsubseteq A[P]_\sigma$,
3. $\forall P' \ : \ ((P' \mid\sim P_\phi) \ and \ (A[P']_\sigma \sqsubseteq A[P]_\sigma)) \Longrightarrow A[P']_\sigma \sqsubseteq A[\partial_{P_\phi}(P)]_\sigma$.

Proof.
1. The process $\partial_{P_\phi}(P)$ is defined so that it can evolve into another process only if the security policy P_ϕ is satisfied.
2. As well as the first property, the proof is obtained directly from the reduction rules of the security operator and the definition of the partial order.
3. Consider a process $P' \in \mathcal{P}$ such that $P' \mid\sim P_\phi$ and $A[P']_\sigma \sqsubseteq A[P]_\sigma$. Suppose $A[P']_\sigma \overset{a}{\rightsquigarrow} A[P'_1]_{\sigma'}$. Since $A[P']_\sigma \sqsubseteq A[P]_{\sigma'}$, we conclude directly from the definition of $\sqsubseteq$ that $A[P]_\sigma \overset{a}{\rightsquigarrow} A[P_1]_{\sigma'}$. Since $P' \mid\sim P_\phi$, we can also conclude from the definition of $\mid\sim$, that $a \mid\sim P_\phi$. Finally, as $A[P]_\sigma \overset{a}{\rightsquigarrow} A[P_1]_{\sigma'}$ and $a \mid\sim P_\phi$ so $A[\partial_{P_\phi}(P)]_\sigma \overset{a}{\rightsquigarrow} A[\partial_{P'_\phi}(P_1)]_{\sigma'}$ and then $A[P']_\sigma \sqsubseteq A[\partial_{P_\phi}(P)]_\sigma$.

Example 3 (Buyer-Seller Protocol). The buyer initiates a communication with the seller and requests for a quote. The seller sends back the quote. If the buyer rejects it then the protocol terminates. Otherwise, the seller sends to the buyer an order confirmation and the buyer confirms his command. In this case, the seller contacts the shipper and asks for delivery details which he transfer to the buyer and the protocol terminates. The security property that we want to apply in this protocol is : "the seller should communicate with the shipper only if the buyer confirms his command". As a consequence, the seller won't send to buyer the delivery details if he has not confirmed his command. The protocol is depicted in Fig.1. Critical actions that we have to supervise are in dark grey in Fig.1.

The choreography's description in global calculus is the following :

$$
\begin{array}{l}
\text{Buyer} \rightarrow \text{Seller} : B2SCh(s). \\
\text{Seller} \rightarrow \text{Buyer} : s\langle ackSession \rangle. \\
\text{Buyer} \rightarrow \text{Seller} : s\langle reqQuote \rangle. \\
\text{Seller} \rightarrow \text{Buyer} : s\langle respQuote, v_{quote}, x_{quote} \rangle. \\
\text{if } resonable(x_{quote})@\text{Buyer then} \\
\quad \text{Buyer} \rightarrow \text{Seller} : s\langle quoteAccept \rangle. \\
\quad \text{Seller} \rightarrow \text{Buyer} : s\langle orderConfirm \rangle. \\
\quad \text{Buyer} \rightarrow \text{Seller} : s\langle ackConfirm \rangle. \\
\quad \text{Seller} \rightarrow \text{Shipper} : B2SHCh(s'). \\
\quad \text{Shipper} \rightarrow \text{Seller} : s'\langle ackSession \rangle. \\
\quad \text{Seller} \rightarrow \text{Shipper} : s'\langle reqDelivDet \rangle. \\
\quad \text{Shipper} \rightarrow \text{Seller} : s'\langle delivDet, v_{delivDet}, x_{delivDet} \rangle. \\
\quad \text{Seller} \rightarrow \text{Buyer} : s\langle delivDet, x_{delivDet}, x_{delivDet} \rangle.0 \\
\text{else} \\
\quad \text{Buyer} \rightarrow \text{Seller} : s\langle quoteReject \rangle.0
\end{array}
$$

The end-point projection gives the end-point behaviors. The buyer's behavior is the following :

$$Buyer[\overline{B2SCh}(\nu s).s \rhd ackSession.\overline{s} \lhd reqQuote.s \rhd respQuote(x_{quote})$$

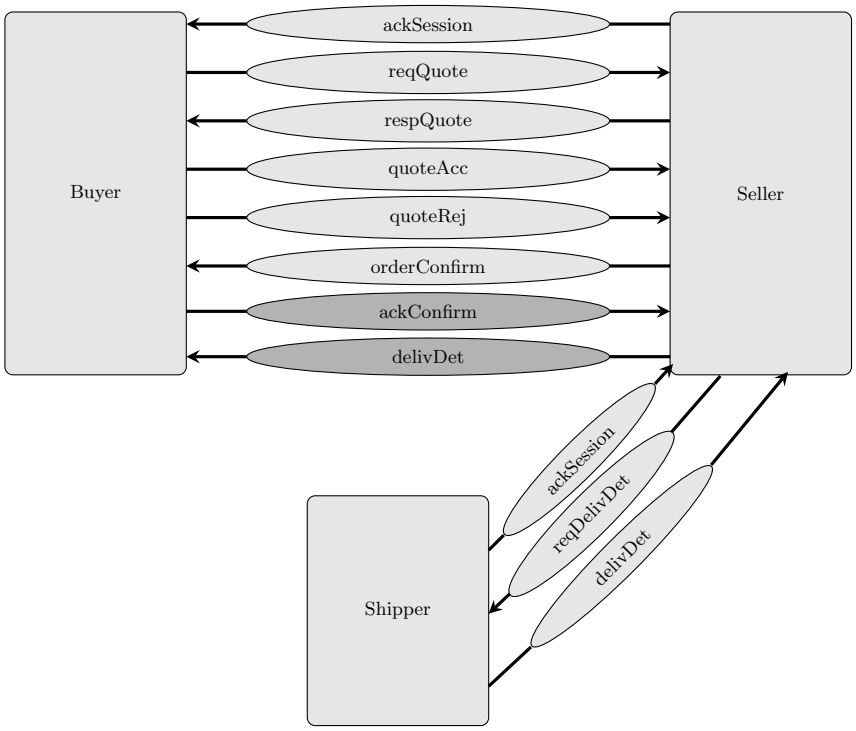


Fig. 1. Buyer Seller Protocol

.if $reasonable(v_{quote})$ then $\overline{s} \lhd quoteAcc.s \rhd orderConfirm.\overline{s} \lhd ackConfirm$

$.s \rhd delivDet(x_{delivDet}).0$ else $\overline{s} \lhd quoteRej.0]$

The seller's behavior is the following :

$$Seller[B2SCh(s).\overline{s} \lhd ackSession.s \rhd reqQuote.\overline{s} \lhd respQuote\langle e_{quote}\rangle.$$

$$(s \rhd quoteAcc.\overline{s} \lhd orderConfirm.s \rhd ackConfirm.$$

$$\overline{B2SHCh}(\nu s').s' \rhd ackSession.\overline{s'} \lhd reqDelivDet\langle e_{buyer}\rangle.s' \rhd delivDet(x_{delivDet})$$

$$.\overline{s} \lhd delivDet(x_{delivDet}).0\ ) + (s \rhd quoteRej.0)]$$

The shipper's behavior is the following :

$$Shipper[B2SHCh(s').\overline{s'} \lhd ackSession.s' \rhd reqDelivDet(x_{buyer})$$

$$.\overline{s'} \lhd delivDet\langle e_{delivDet}\rangle.0]$$

The security property has the following behavior :

$$
\begin{aligned}
P_\phi = \text{rec } X.(\ & \overline{s} \lhd \bigoplus_{op \neq delivDet} op_i\langle e\rangle.X \quad \oplus \\
& s \rhd \sum_{op \neq ackConfirm} op_i(x_i).X \quad \oplus \\
& s \rhd ackConfirm.\text{rec } Y.(\overline{s'} \lhd \bigoplus_i op_i\langle e\rangle.Y \oplus s' \rhd \sum_i op_i(x_i).Y \\
& \oplus \overline{s} \lhd \bigoplus_i op_i\langle e\rangle.Y) \\
)\ &
\end{aligned}
$$

The security property P_ϕ is written using the recursion operator. In the recursion block, there are 3 processes combined with the internal choice. The first process is : $\overline{s} \lhd \bigoplus_{op \neq delivDet} op_i\langle e\rangle.X$ which says that, through the session channel s (shared between the buyer and the seller), the buyer can execute any sending action unless delivery details which should be done after the seller has received a confirmation from the buyer. The second process is : $s \rhd \sum_{op \neq ackConfirm} op_i(x_i).X$ which says that the seller can receive any operation other than $ackConfirm$. This action is intercepted in the third process $s \rhd ackConfirm.\text{rec } Y.(\overline{s'} \lhd \bigoplus_i op_i\langle e\rangle.Y \oplus s' \rhd \sum_i op_i(x_i).Y \oplus \overline{s} \lhd \bigoplus_i op_i\langle e\rangle.Y)$. In this process, after the seller has received a confirmation from the buyer, he can communicate with the shipper through the session channel s' without any restriction and he has also no restriction on his sending actions to the buyer, so he can send him the delivery details.

6 Related Works

Web services verification have been a subject of interest of several research efforts. Some of the relevant contributions in this domain are cited in this section. Most of formal approaches introduced a monitor which does not stop the program when a violation is detected. Moreover, these contributions implement a monitor as a web service in addition to other web services. The originality of our work is the introduction of the monitor within concerned participants processes. In [11], a run-time event-based approach to deal with the problem of monitoring conformance of interaction sequences is presented. When a violation is detected, the program shows errors in dashboards. In [12], authors introduce an approach to verify the conformance of a web service implementation against a behavioral specification, through the application of testing. The Stream X-machines are used as an intuitive modeling formalism for constructing the behavioral specification of a stateful web service and a method for deriving test cases from that specification in an automated way. The test generation method produces complete sets of test cases that, under certain assumptions, are guaranteed to reveal all non-conformance faults in a service implementation under test. However, this approach only returns non-conformance faults and does not react dynamically against these errors. While, in [13], authors propose a monitoring framework of a choreographed service which supports the early detection of faults and decide

whether it is still possible to continue the service. Authors in [**?**] have proposed service automata as a framework for enforcing security policies in distributed systems. They encapsulate the program in a service automaton composed of the monitored program, an interceptor, an enforcer, a coordinator and a local policy. The interceptor intercepts critical actions and passes them to the coordinator that determines whether the action complies the security policy or not and decides upon possible countermeasures then the enforcer implements these decisions. However the authors do not precise how to detect critical actions.

7 Conclusion

In this paper, we have introduced a formal approach allowing to automatically enforce a security policy on choreographed services. Indeed, we introduced a new calculus with an enforcement operator ∂_{P_ϕ}. The semantics of the proposed calculus insure that choreographed services can evolve only if it does not violate the enforced security policy. The originality of our work consists on the fact that we do not add a new web service as a monitor but rather we wrap the security policy inside the choreographed services.

Future work will focus on the definition of a complete mapping between WS-CDL and global calculus. Moreover, we will seek means to optimize the enforced choreographed services so that we reduce as much as we can the overhead due to the enforcement operator.

References

1. Corporation, I.: Business process execution language for web services bpel-4ws. http://www.ibm.com/developerworks/library/ws-bpel/ (2002)
2. F.Leymann: Web services flow language (wsfl) version 1.0. Technical Report, International Business Machines Corporation (IBM) (May 2001)
3. Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y.: Web services choreography description language version 1.0. W3C Working Draft (December 2004)
4. Langar, M., Mejri, M., Adi, K.: Formal enforcement of security policies on concurrent systems. J. Symb. Comput. **46**(9) (2011) 997–1016
5. Khoury, R., Tawbi, N.: Corrective enforcement: A new paradigm of security policy enforcement by monitors. ACM Trans. Inf. Syst. Secur. **15**(2) (2012) 10
6. Yi, X., Kochut, K.: A cp-nets-based design and verification framework for web services composition. In: ICWS. (2004) 756–760
7. Hamadi, R., Benatallah, B.: A petri net-based model for web service composition. In: ADC. (2003) 191–200
8. Salaün, G., Bordeaux, L., Schaerf, M.: Describing and reasoning on web services using process algebra. In: ICWS. (2004) 43–
9. Kazhamiakin, R., Pandya, P.K., Pistore, M.: Timed modelling and analysis in web service compositions. In: ARES. (2006) 840–846
10. Carbone, M., Honda, K., Yoshida, N.: Structured communication-centred programming for web services. In: ESOP. (2007) 2–17

11. Baouab, A., Perrin, O., Godart, C.: An optimized derivation of event queries to monitor choreography violations. In: ICSOC. (2012) 222–236
12. Dranidis, D., Ramollari, E., Kourtesis, D.: Run-time verification of behavioural conformance for conversational web services. In: ECOWS. (2009) 139–147
13. Ardissono, L., Furnari, R., Goy, A., Petrone, G., Segnan, M.: Monitoring choreographed services. In: Innovations and Advanced Techniques in Computer and Information Sciences and Engineering. (2007) 283–288
14. Fu, X., Bultan, T., Su, J.: Analysis of interacting bpel web services. In: WWW. (2004) 621–630