Enforcing Security Policies on Choreographed Services using Rewriting Techniques

Karim Dahmani
karim.dahmani@fst.rnu.tn

Mahjoub Langar
mahjoub.langar@ift.ulaval.ca

LIP2 Research Laboratory
Faculté des Sciences de Tunis
Tunis, Tunisia

***Abstract*—This paper presents an automated formal approach for enforcing security policies on a choreography of Web Services. We take as input a formal description of a choreography of web services and a security property represented by a process, then we define some rewriting rules and rewrite the two processes in order to make them synchronize on each communication action. This approach produces as output a secure version of the concerned web service which behaves like the original one but does not violate the security property.**

***Keywords*-Web Service Composition Security; Instrumentation; Choreography; Formal Verification; End-Point Calculus.**

I. INTRODUCTION

Web Services (WS) are distributed and modular applications that communicate by message passing in order to complete specific activities. Composition of WS consists in combining different WS to provide value-added services. WS 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 WS choreography, does not assume the exploitation of a central coordinator but rather defines complex tasks via the definition of the conversation that should be undertaken by each participant. Several proposals exist for orchestration and choreography languages such as Business Process Execution Language (BPEL) [1] for orchestration and Web Service Choreography Description Language (WS-CDL) [2] for choreography. Since the orchestration technique uses a central coordinator that composes the available services, it seems trivial to enforce security policies. So the technique that will be used in this paper for composing WS is the choreography. One of the main challenges for researchers in this domain is the formalization of these composition languages. Although, several contributions have been developed in the last decade that formalize WS-CDL such as the Global Calculus (GC) and the End-Point Calculus (EPC) proposed by Carbone et al. [3], the Choreography Language proposed by N. Busi et al. [4], The Choreography Description Language proposed by H. Yang et al. [12] and timed automata proposed by G. Diaz et al. [5]. The formal specification language used in this paper is the End-Point Calculus that has been introduced by Carbone et al. [3]. One of the reasons behind this choice is that the end-point calculus is a modified version of the pi-calculus, so its syntax is more familiar and its expressivity is stronger.

The need of secure WS composition has led to a great interest from researchers in the last decade. In this paper, we propose an automated formal approach for the enforcement of security policies on choreographed services. We take as input a formal description of the behavior of a participant in a choreography and a security property. We define rewriting rules for adding some special actions to processes and security properties in order to ensure synchronization and consequently control the evolution of the behavior of a participant.

This paper is structured as follows: in Section II, we introduce the choreography specification language used in this topic. In Section III we present the security property specification language. Section IV deals with the enforcement approach. The proof of this approach is given in Section V. Related work is in Section VI and conclusion and future work are in Section VII.

II. CHOREOGRAPHY SPECIFICATION LANGUAGE

Carbone et al. [3] have proposed a formal language for specifying a choreography of WS. This language is the GC. It describes behaviors of WS from a global viewpoint. GC is distilled from WS-CDL. Carbone et al. [3] have also proposed a second formal language: the EPC, which specifies behaviors of WS from a local viewpoint. Finally, a projection under some assumptions from GC to EPC have been proposed by Carbone et al. [3], which is called the End-Point Projection (EPP). The language adapted in this paper for formally specifying processes and security properties is EPC.

A. Syntax of the End-Point Calculus

EPC describes the behavior of each participant in the choreography from its end-point view. EPC is a variant of the pi-calculus augmented with the notion of participants and their local states. We present hereafter the syntax and formal semantics of EPC where P,Q range over processes, ch range over service channels, s range over session channels, op_i range over operator names, x range over

 ISBN: 978-1-61208-376-6

variables, *e* range over expressions and *X* range over term variables.

$$P ::= !ch(\tilde{s}).P \mid \overline{ch}(\nu\,\tilde{s}).P \mid s \triangleright \Sigma_i op_i(x_i).P_i \mid \overline{s} \triangleleft op(e).P \mid x := e.P \mid P \oplus Q \mid P|Q \mid if\ e\ then\ P\ else\ Q \mid (\nu\, s)P \mid rec\ X.P \mid 0$$

- $!ch(\tilde{s}).P$ and $\overline{ch}(\nu\,\tilde{s}).P$ represent session initiation. $!ch(\tilde{s}).P$ is used for input and $\overline{ch}(\nu\,\tilde{s}).P$ for output. $!ch(\tilde{s}).P$ 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 $s \in \tilde{s}$. $\overline{ch}(\nu\,\tilde{s}).P$ is an invocation of a service located at the service channel *ch* and an initiation of a communication session that will occur through session channels $s \in \tilde{s}$. After a session has been initiated between two participants and freshly generated session channels have been shared between them, they can communicate via these channels using the communication constructs.
- $s \triangleright \Sigma_i op_i(x_i).P_i$ is an offer of one of operators op_i and a reception of an expression *e* through the session channel *s* that will be evaluated and stored in the local variable *x*. A participant having this behavior will receive an invocation of one of its operator names op_i and an expression *e*. The value of *e* is saved in its local variable *x*.
  For instance, a seller service receives a confirmation or a cancellation for a purchase :
  $s \triangleright confirmPurchase(x_1).P + s \triangleright cancelPurchase(x_2).0$
- $\overline{s} \triangleleft op(e).P$ sends the expression *e* and invokes operator *op* through the session channel *s*. Indeed, a buyer service requests a quote of a chosen product from a seller through the channel *s*:
  $\overline{s} \triangleleft quoteRequest(e_{product}).P$
  In addition, operator names $op_1, op_2, \ldots$ are invoked by a message sender or offered by a message receiver. Operator names in in-session communications are analogous to methods in objects [3].
- $x := e.P$ is the assignment operator. It is a local operation. It assigns the result of the evaluation of the expression *e* to the variable *x*. For example, the buyer assigns to its variable *x* the value of the quote received from the seller : $x_{quote} := e_{quote}.P$
- *if e then P else Q* is a choice based on the evaluation of the boolean expression *e*. For example, the buyer accepts the quote if it is under 1000
  $if\ e_{quote} < 1000\ then\ \overline{s} \triangleleft accept(e_{quote}).P\ else\ \overline{s} \triangleleft reject(e_{quote}).0$
- $P \oplus Q$ is the non deterministic choice. When the choice of the buyer is arbitrary, it would be written as: $\overline{s} \triangleleft accept(e_{quote}).P \oplus \overline{s} \triangleleft reject(e_{quote}).0$
- $P|Q$ is the parallel composition of processes. For example, a seller that offers his service to buyers should have his service running in parallel for new requesters $!ch_{seller}(s).P|P'|P''|...$ where *P',P'',...* are processes dealing with different buyers.
- $(\nu\, s)P$ expresses the fact that the session channel *s* is local to *P*. It is used to restrict a session channel to be used between only two participants that communicate through it.
- *rec X.P* is the recursion operator used to express repetitive behaviors. For example, a participant having the following behavior will always request quotes until he receives an acceptation $rec\ X.\overline{s} \triangleleft quoteRequest(e). s \triangleright accept.P \oplus s \triangleright reject.X$.
- Finally, 0 is the inaction.

Processes are located within participants. Participants and their composition are called Networks (written *N,M,...*), whose grammar is given by:

$$N ::= A[P]_\sigma \mid N|M \mid (\nu\, s)N \mid \epsilon \quad .$$

For more details about the syntax of EPC, the reader can refer to [3].

B. Semantics of the End-Point Calculus

In order to minimize the number of reduction rules, we define $\equiv$ as the least congruence generated from:

$$P|0 \equiv P$$
$$P|Q \equiv Q|P$$
$$(P|Q)|R \equiv P|(Q|R)$$
$$P \oplus P \equiv P$$
$$P \oplus Q \equiv Q \oplus P$$
$$(P \oplus Q) \oplus R \equiv P \oplus (Q \oplus R)$$
$$(\nu\, s)0 \equiv 0$$
$$(\nu\, s_1)(\nu\, s_2)P \equiv (\nu\, s_2)(\nu\, s_1)P$$
$$((\nu\, s)P)|Q \equiv (\nu\, s)(P|Q) \quad (s \notin fn(Q))$$

$$A[P]_\sigma \equiv A[Q]_\sigma \quad (P \equiv Q)$$
$$A[(\nu\, s)P]_\sigma \equiv (\nu\, s)(A[P]_\sigma)$$
$$M|\epsilon \equiv M$$
$$M|N \equiv N|M$$
$$(M|N)|L \equiv M|(N|L)$$
$$(\nu\, s)\epsilon \equiv \epsilon$$
$$(\nu\, s_1)(\nu\, s_2)M \equiv (\nu\, s_2)(\nu\, s_1)M$$
$$((\nu\, s)M)|N \equiv (\nu\, s)(M|N) \quad (s \notin fn(N))$$

The operational semantics of EPC are given in Figure 1.

- Init 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* using the binding operator ν .

- Comm explains how a communication is established between two participants: when *B* invokes the operator op_j, which is offered by *A*, and sends the expression e_j, which will be evaluated to value *v* at *A*, then *A* receives it and assigns *v* to its local variable x_j.
- Assign is a local operation. Assignment rule evaluates an expression *e* and assigns the result of this evaluation to variable *x* in *A*, then *A* behaves as *P*.
- IfThenElse evaluates the boolean expression *e* and following the result of this evaluation, it behaves either as P_1 or P_2.
- Par shows the behavior of two concurrent processes.
- Sum shows the alternative choice behavior.
- Rec says that if the process *P*, within which we replace each occurrence of *X* by *rec X.P*, behaves as *P'* then *rec X.P* will behave as *P'*.
- Res restricts the use of session channels $\tilde{s}$ to the process *P* in *A*.

Finally, the following rule says we take the reductions up to the structural rule:

$$\frac{M \equiv M' \quad M' \to N' \quad N' \equiv N}{M \to N} \; Struct - NW$$

C. Example

We consider a simplified version of a travel reservation system. The scenario consists of three participants: a traveler, a travel agent and an airline reservation system. The traveler is planning for taking a trip. Once the traveler selects a trip, he submits his choice to the travel agent. The travel agent checks for seats availability within the airline reservation system and sends back either a trip cancel or a validation. The traveler wants to reserve tickets for this trip by sending payment details to the travel agent. The travel agent now must verify one more time availability of seats. If the seats are still available then the airline reservation system accepts the payment details and sends back to the travel agent tickets of the trip. The travel agent responds to the traveler either by tickets of the trip or by canceling the reservation.

The behavior of the traveler is given in EPC by:

$$\overline{ch_{TA}}(\nu s).s \triangleright ack.\bar{s} \triangleleft orderTrip(e_1).$$
$$s \triangleright cancel.0 \oplus s \triangleright available(x_1).\bar{s} \triangleleft book(e_2).$$
$$s \triangleright cancelBook.0 \oplus s \triangleright tickets(x_2).0$$

The traveler starts by opening a session with the travel agent through the public service channel ch_{TA} and initiates a communication channel *s* through which the communication between the traveler and the travel agent will occur. Then, the traveler receives through *s* an acknowledgment message $s \triangleright ack$. After that, the traveler sends an order trip $\bar{s} \triangleleft orderTrip(e_1)$ with expression e_1 which contains details about the chosen trip. At this point, there are two scenarios: the traveler either receives a cancel request $s \triangleright cancel$ when there are no available seats, or an available message $s \triangleright available(x_1)$ containing details of the trip. In this case, the traveler may book the flight $\bar{s} \triangleleft book(e_2)$. Finally, traveler receives *tickets* message $s \triangleright tickets(x_2)$ if the transaction has succeed, otherwise he will receive a *cancelBook* message .

The behavior of the travel agent is given by:

$$!ch_{TA}(s).\bar{s} \triangleleft ack.s \triangleright orderTrip(x_1).$$
$$\overline{ch_A}(\nu s').s' \triangleright ack.\overline{s'} \triangleleft checkSeat(e_1).$$
$$s' \triangleright noSeats.\bar{s} \triangleleft cancel.0 \oplus$$
$$s' \triangleright seatsOK(x_2).\bar{s} \triangleleft available(e_2).s \triangleright book(x_3).$$
$$\overline{s'} \triangleleft reserve(e_3).$$
$$s' \triangleright reserved(x_4).\bar{s} \triangleleft tickets(e_4).0 \oplus$$
$$s' \triangleright notReserved(x_5).\bar{s} \triangleleft cancelBook.0$$

The travel agent offers his service through ch_{TA} by providing a communication channel s. Once his service is invoked, he sends an acknowledgment message, receives an order trip, then contacts the airline service through its public service channel ch_A. The communication between the airline service and the travel agent occurs through the session channel s'. The travel agent looks for available seats $\bar{s}' \triangleleft checkSeat(e_1)$. If there are no available seats then he receives *noSeats* message $s' \triangleright noSeats$ and he sends a *cancel* message $\bar{s} \triangleleft cancel$ to the traveler. Otherwise he receives a *seatsOK* message $s' \triangleright seatsOK(x_2)$ and sends an *available* message $\bar{s} \triangleleft available(e_2)$ to the traveler. After that, the travel agent receives a *book* message from the traveler $s \triangleright book(x_3)$ and proceeds to flight reservation $\bar{s}' \triangleleft reserve(e_3)$. Depending on seats availability, he receives either a confirmation message $s' \triangleright reserved(x_4)$ or a *notReserved* message $s' \triangleright notReserved(x_5)$. In the first case, he sends tickets $\bar{s} \triangleleft tickets(e_4)$ to the traveler elsewhere he sends a book cancellation $\bar{s} \triangleleft cancelBook.$

The behavior of the airline is given by:

$$!ch_A(s').\overline{s'} \triangleleft ack.s' \triangleright checkSeat(x_1).$$
$$if \; available(x_1) \; then \; \overline{s'} \triangleleft seatsOk(e_1).$$
$$s' \triangleright reserve(x_2).if \; available(x_2) \; then$$
$$\overline{s'} \triangleleft reserved(e_2).0 \; else$$
$$\overline{s'} \triangleleft notReserved(e_3).0$$
$$else \; \overline{s'} \triangleleft noSeats.0$$

The airline service offers his service through the service channel ch_A. Once his service is invoked, he sends an acknowledgment message. Then, he receives a *checkSeat* request. Subject to availability, he responds with a *seatsOK* or *noSeats* message. In the first case, he may receive a seat's booking request $s' \triangleright reserve(x_2)$. At this stage, the airline service checks another time seats availability before finalizing the process by either sending a *reserved* $s' \triangleleft reserved(e_2)$ or $\overline{s'} \triangleleft notReserved(e_3)$ as *notReserved* message.

III. Security Policy Specification Language

In this Section, we introduce the Security Policy Calculus (SPC), a formalism used for describing security policies. SPC is considered as a subset of EPC in the sense that it uses only some operators of EPC. Indeed the operators that are

used in SPC are communication actions, recursion, indeterministic choice and no action. Security policies will be represented by processes that will monitor the execution of an another process from EPC.

security properties that we verify in this paper are safety properties and liveness properties without infinite behavior. The reason behind this choice is that some liveness properties can only be verified statically.

B. Shortcuts

$$\frac{}{\square}\ \text{Init}$$

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

$$\frac{\sigma_A \vdash e \Downarrow v}{A[s \triangleright \Sigma_i op_i(x_i).P_i|P']_{\sigma_A}|B[\overline{s} \triangleleft op_j\langle e\rangle.Q|Q']_{\sigma_B} \rightarrow A[P_j|P']_{\sigma_A[x_j \mapsto v]}|B[Q|Q']_{\sigma_B}}$$

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

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

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

$$\frac{A[P_1|P']_{\sigma_A} \rightarrow A[P_1'|P']_{\sigma_A'}}{A[P_1|P_2|P']_{\sigma_A} \rightarrow A[P_1'|P_2|P']_{\sigma_A'}}\ \text{Par} \qquad \frac{A[P_1|P']_{\sigma_A} \rightarrow A[P_1'|P']_{\sigma_A'}}{A[P_1 \oplus P_2|P']_{\sigma_A} \rightarrow A[P_1'|P']_{\sigma_A'}}\ \text{Sum}$$

$$\frac{A[P[\text{rec } X.P/X]]_{\sigma_A} \rightarrow A[P']_{\sigma_A'}}{A[\text{rec } X.P]_{\sigma_A} \rightarrow A[P']_{\sigma_A'}}\ \text{Rec} \qquad \frac{A[P]_{\sigma_A} \rightarrow A[P']_{\sigma_A'}}{A[(\nu s)P]_{\sigma_A} \rightarrow A[(\nu s)P']_{\sigma_A'}}\ \text{Res}$$

Figure 1

A. Syntax

The syntax of SPC is given by:

$\varphi ::= \overline{s} \triangleleft \oplus op_i.\varphi_i \mid s \triangleright \Sigma op_i.\varphi_i \mid \varphi_1 \oplus \varphi_2 \mid rec\ X.\varphi \mid 0$

- The construct $\overline{s} \triangleleft \oplus op_i.\varphi_i$ expresses the fact that invoking one of operators op_i through session channel *s* is permitted by φ.
- Next we have $s \triangleright \Sigma op_i.\varphi_i$, which allows reception of operators op_i through *s*.
- The indeterministic branching is given by $\varphi_1 \oplus \varphi_2$.
- For representing repeated behaviors, we use the recursion operator and
- finally, 0 denotes the lack of actions.

For describing security properties, we need usually to express the prohibition of executing some actions. In our case, when we want for example to interdict sending operation op_1 through *s* we would write this security property: $\varphi = \overline{s} \triangleleft \underset{i \neq 1}{\oplus} op_i.0$, which says that we can invoke anyone of the operators op_i unless op_1. If we want this behavior to be repeatedly verified we would write $\varphi = rec\ X.\overline{s} \triangleleft \underset{i \neq 1}{\oplus} op_i.X$. The semantics are the same as for EPC since SPC is a subset of EPC.

Usually, we use temporal logics for describing security properties but when using the security policy calculus we also reach our goal of expressing any security property that we want to enforce on the behavior of a WS. Since this approach introduces a dynamic verification of WS, so the

For shortness, we will denote by ϕ_s and $\phi_{\overline{s}}$ the portions of a security property that respectively allows all input (output) interactions through *s*. So $\phi_s = s \triangleright \Sigma op_i(x_i)$ and $\phi_{\overline{s}} = \overline{s} \triangleleft \oplus op_i(e_i)$.

C. Example

In the airline reservation system, it is assumed the travel agent wants to be sure that his service does not send tickets before the reception of payment details. The travel agent receives payment details within the book message $s \triangleright book(x_3)$ and sends tickets within the tickets message $\overline{s} \triangleleft tickets(e_4)$. So we want to ensure that $\overline{s} \triangleleft tickets(e_4)$ does not occur before $s \triangleright book(x_3)$. The security property will be written as follows:

$$rec\ X.\overline{s} \triangleleft \underset{op_i \neq tickets}{\oplus} op_i(e_i).X \oplus s \triangleright \underset{op_i \neq book}{\Sigma} op_i(x_i).X \oplus \phi_{s'}.X \oplus \phi_{\overline{s'}}.X \oplus s \triangleright book(x).(rec\ Y.\phi_s.Y \oplus \phi_s.Y \oplus \phi_{\overline{s'}}.Y \oplus \phi_{s'}.Y)$$

The security property is written using a recursion. The idea is to put after each action different from *book* and *tickets* the recursion variable *X*. Thus, the property will remain invariant when executing these actions. It will evolve only by executing the book message. In this recursion, one of these 4 blocks will be executed:

- $\overline{s} \triangleleft \underset{op_i \neq tickets}{\oplus} op_i(e_i).X$: it prohibits invoking *tickets* operator through *s*, which is shared between the traveler and the travel agent. Any message different from *tickets* can be sent.

- $s \triangleright \sum_{op_i \neq book} op_i(x_i).X$: this block prohibits the reception of *book* operator invocation through the session channel *s*. Any message different from *book* can be received.
- $\phi_{s'}.X$ and $\phi_{\overline{s'}}.X$: all actions are permitted between the travel agent and the airline reservation service. The channel *s'* is a session channel shared between the travel agent and the airline reservation system.
- $s \triangleright book(x).rec\ Y.\phi_{\overline{s}}.Y \oplus \phi_s.Y \oplus \phi_{\overline{s'}}.Y \oplus \phi_{s'}.Y$ this block intercepts the invocation of the *book* operator within the travel agent and then we take off the control on *tickets* operator by allowing all actions between the traveler and the traveler agent through *s* and the travel agent and the airline reservation system through *s'* to be executed.

IV. Enforcement Approach

In this Section, we will introduce our enforcement approach using rewriting techniques. This approach consists in adding some special actions to processes representing the behavior of a WS and a security property in order to make them synchronize on each interaction that will occur.

A. Communication Actions of EPC

Communication actions are used in this context to designate interactions of EPC. Interactions of EPC are given by these two constructs: $\overline{s} \triangleleft op(e)$ and $s \triangleright \Sigma op_i(x_i)$. They are distinguished by three criteria:

- session channel (*s*),
- operator name (op_i),
- and direction ($\overline{s} \triangleleft$ or $s \triangleright$).

Indeed, each interaction in EPC occurs through a session channel that have been freshly generated and shared between two participants. Within each interaction an operator is either invoked or offered depending on the direction of the interaction. For instance, $\overline{s} \triangleleft op_1(e)$ is an invocation of the operator op_1 through the session channel *s*, $s' \triangleright op_2(x)$ is a reception of an invocation of the operator op_2 on the session channel *s'*. The goal of this approach is to monitor the execution of interactions inside a choreography. This goal will be achieved by controlling the execution of communication actions of EPC.

B. Synchronization Actions

Synchronization actions are special actions that we add to the process and to the security property enforced on this process in order to ensure the interception of each communication action by the monitor. The idea of using synchronization actions to intercept actions is inspired from [6]. So, given a communication action $\overline{s} \triangleleft op(e)$ (respectively $s \triangleright \Sigma op_i(x_i)$), the corresponding synchronization action is $\overline{\overline{s} \triangleleft op(e)}$ (respectively $\overline{s \triangleright \Sigma op_i(x_i)}$).

C. Rewriting Processes

In order to achieve our goal that consists on enforcing a security property on the behavior of a participant *A* in a choreography, we need to rewrite its process by adding synchronization actions. Informally, we will add before each communication action its corresponding synchronization action. Formal rules for rewriting a process *P* of EPC are:

$$\langle !ch(\tilde{s}).P \rangle := !ch(\tilde{s}).\langle P \rangle$$
$$\langle \overline{ch}(\nu \tilde{s}).P \rangle := \overline{ch}(\nu \tilde{s}).\langle P \rangle$$
$$\langle x:=e.P \rangle := x:=e.\langle P \rangle$$
$$\langle P \oplus Q \rangle := \langle P \rangle \oplus \langle Q \rangle$$
$$\langle P|Q \rangle := \langle P \rangle | \langle Q \rangle$$
$$\langle if\ e\ then\ P\ else\ Q \rangle := if\ e\ then\ \langle P \rangle\ else\ \langle Q \rangle$$
$$\langle rec\ X.P \rangle := rec\ X.\langle P \rangle$$
$$\langle \overline{s} \triangleleft op(e).P \rangle := \overline{\overline{s} \triangleleft op(e)}.\overline{s} \triangleleft op(e).\langle P \rangle$$
$$\langle s \triangleright op(x).P \rangle := \overline{s \triangleright op(x)}.s \triangleright op(x).\langle P \rangle$$

D. Rewriting Security Properties

Rewriting the security property consists on replacing each communication action by its synchronization action. Formal rules for rewriting security properties are:

$$\langle \overline{s} \triangleleft \oplus op_i.\varphi_i \rangle := \overline{\overline{s} \triangleleft \oplus op_i}.\langle \varphi_i \rangle$$
$$\langle s \triangleright \Sigma op_i.\varphi_i \rangle := \overline{s \triangleright \Sigma op_i}.\langle \varphi_i \rangle$$
$$\langle \varphi_1 \oplus \varphi_2 \rangle := \langle \varphi_1 \rangle \oplus \langle \varphi_2 \rangle$$
$$\langle rec\ X.\varphi \rangle := rec\ X.\langle \varphi \rangle$$

E. Restriction Operator

In order to make the rewritten security property φ and the rewritten process *P* synchronize, we will define an operator of EPC that we call the restriction operator and we denote by $P \backslash \varphi$. The role of this operator is to let the process evolve normally when no communication actions is willing to occur. Before a communication action will occur $P \backslash \varphi$ will intercept its synchronization action and verify if the security property can evolve by executing this synchronization action. If it is the case then *P* and φ execute this synchronization action. Else *P* will block and will not execute any other actions. An another role of this enforcement operator is that it hides synchronizations of *P* and φ for the rest of the choreography. Thus, executions of synchronization actions in EPC will be marked by τ as silent actions. Thus our restriction operator does not affect the evolution of *P* when no synchronization action is willing to occur. $P \backslash \varphi$ must ensure the synchronization of *P* and φ on only synchronization actions.

F. Normal Form of a Process

Every process representing the local behavior of a participant in a WS can be written as an internal sum of processes, which we call the normal form of a process:

$P = \bigoplus_{i \in I} a_i.P_i$ where a_i range over atomic actions, *I* is a finite subset of natural numbers, and P_i range over processes.

Atomic actions of EPC are: session initiation request

($\overline{ch}(\nu\,\tilde{s})$), session initiation offer ($!ch(\tilde{s})$), communication input ($s \triangleright op(x)$), communication output ($\overline{s} \triangleleft op(e)$), assignment ($x := e$) and synchronization actions ($\overline{s \triangleright op(x)}$, $\overline{\overline{s} \triangleleft op(e)}$).

G. Simulation

We say that a process P can execute an action a and becomes P' and we write $P \xrightarrow{a} P'$ if, when we write P in its normal form ($P = \oplus_{i \in I} a_i . P_i$), there exists $j \in I$ such that $a_j = a$ and $P_j \equiv P'$ where $\equiv$ is the structural equality defined in the semantics of EPC.

H. Semantics

Reduction rules for making $P \backslash \varphi$ progress when executing synchronization actions are given by:

$$\frac{P \xrightarrow{\overline{\overline{s} \triangleleft op(e)}} P' \quad \varphi \xrightarrow{\overline{\overline{s} \triangleleft op(e)}} \varphi'}{A[P \backslash \varphi]_\sigma \xrightarrow{\tau} A[P' \backslash \varphi']_\sigma} \qquad \frac{P \xrightarrow{\overline{s \triangleright op(x)}} P' \quad \varphi \xrightarrow{\overline{s \triangleright op(x)}} \varphi'}{A[P \backslash \varphi]_\sigma \xrightarrow{\tau} A[P' \backslash \varphi']_\sigma}$$

These rules say that each synchronization action of P will be intercepted by φ and it cannot be executed if φ prohibits it. If the synchronization action can be executed by φ then P becomes silently P' and φ becomes φ' .

I. Example

Consider the airline reservation system case study. We will enforce the security property φ defined in the precedent example on the behavior P of the travel agent defined in the first example of this paper. The rewritten process and security property are:

$$P = !ch_{TA}(s).\overline{\overline{s} \triangleleft ack}.\overline{s} \triangleleft ack.\overline{s \triangleright orderTrip(x_1)}.$$
$$s \triangleright orderTrip(x_1).\overline{ch_A}(\nu\, s').\overline{s' \triangleright ack}.s' \triangleright ack.$$
$$\overline{\overline{s'} \triangleleft checkSeat(e_1)}.\overline{s'} \triangleleft checkSeat(e_1).$$
$$(\overline{s' \triangleright noSeat}.s' \triangleright noSeat.\overline{\overline{s} \triangleleft cancel}.\overline{s} \triangleleft cancel.0)$$
$$\oplus\, \overline{s' \triangleright seatOk(x_2)}.s' \triangleright seatOk(x_2).\overline{\overline{s} \triangleleft available(e_2)}.$$
$$\overline{s} \triangleleft available(e_2).\overline{s \triangleright book(x_3)}.s \triangleright book(x_3).$$
$$\overline{\overline{s'} \triangleleft reserve(e_3)}.\overline{s'} \triangleleft reserve(e_3).\overline{s' \triangleright reserved(x_4)}.$$
$$s' \triangleright reserved(x_4).\overline{\overline{s} \triangleleft ticket(e_4)}.\overline{s} \triangleleft ticket(e_4).0$$
$$\oplus\, \overline{s' \triangleright notReserved(x_5)}.s' \triangleright notReserved(x_5).$$
$$\overline{\overline{s} \triangleleft cancelBook}.\overline{s} \triangleleft cancelBook.0$$

$$\varphi = rec\ X.\overline{\overline{s} \triangleleft \oplus_{op_i \neq ticket} op_i(e_i)}.X \oplus \overline{s \triangleright \sum_{op_i \neq book} op_i(x_i)}.X \oplus$$
$$\langle \phi_{s'} \rangle . X \oplus \langle \phi_{\overline{s'}} \rangle . X \oplus \overline{s \triangleright book(x)}.rec\ Y.\langle \phi_s \rangle . Y \oplus \langle \phi_s \rangle . Y \oplus$$
$$\langle \phi_{\overline{s'}} \rangle . Y \oplus \langle \phi_{s'} \rangle . Y$$

where $\langle \phi_s \rangle = \overline{s \triangleright \sum_i op(x)}$, $\langle \phi_{\overline{s}} \rangle = \overline{\overline{s} \triangleleft \oplus_i op_i(e_i)}$ and similarly for $\langle \phi_{s'} \rangle$ and $\langle \phi_{\overline{s'}} \rangle$.

$TravelAgent[P \backslash \varphi]$ will use first Init reduction rule to open a parallel session then for each communication action, it will synchronize with φ using the reduction rules of $P \backslash \varphi$ and then communicates using communication reduction rules. We can see easily that this process P satisfies the security property φ .

V. Proof of the Approach

In this Section, we prove the correctness of our theory by defining first a partial order over processes and the satisfaction notion.

A. Definition (Subprocess)

Let P, Q be two processes. We say that P is a subprocess of Q and we write $P \subseteq Q$ if the following condition hold : $P \xrightarrow{a} P' \Rightarrow Q \xrightarrow{a} Q'$ and $P' \subseteq Q'$.

B. Definition (Safe Action, Safe Trace)

A trace ξ of EPC is a sequence of atomic actions executed by a process. An atomic action is said to be *safe* if it is not a synchronization action. A trace is said to be safe if it contains only safe actions.

C. Definition (Progression of P)

We say that a process P can progress by executing some safe actions and a synchronization action a and become Q, and we write $P \xrightarrow{a} \!\!| Q$ if it exists a safe trace ξ and a process P' such that $P \xrightarrow{\xi} P'$ and $P' \xrightarrow{a} Q$.

D. Definition (Satisfaction Notion)

We say that a process P satisfies a security property φ and we write $P \approx \varphi$ if for all synchronization action a such that $P \xrightarrow{a} \!\!| P'$ we have $\varphi \xrightarrow{a} \varphi'$ and $P' \approx \varphi'$.

E. Theorem

Let P be a process and φ a security property. The following properties hold :

- $P \backslash \varphi \subseteq P$,
- $P \backslash \varphi \approx \varphi$,
- $\forall P' \approx \varphi,\ P' \subseteq P \Rightarrow P' \subseteq P \backslash \varphi$.

F. Proof

- The proof is obtained directly from the reduction rules of our enforcement operator and from the definition of $\subseteq$. Indeed $P \backslash \varphi$ is defined so that it cannot execute any actions that P does not execute it.
- Let a be a synchronization action such that $P \backslash \varphi \xrightarrow{a} \!\!| P' \backslash \varphi'$. It exists a safe trace ξ such that $P \backslash \varphi \xrightarrow{\xi} P'' \backslash \varphi$ and $P'' \backslash \varphi \xrightarrow{a} P' \backslash \varphi'$. But executions of synchronization actions by $P'' \backslash \varphi$ are given by In-Sync and Out-Sync rules. Then we have necessarily $\varphi \xrightarrow{a} \varphi'$.
- Let P' be a process satisfying a security property φ such that $P' \subseteq P$. Suppose $P' \xrightarrow{a} P''$. As $P' \subseteq P$ then $P \xrightarrow{a} Q$. If a is a synchronization action then from the hypothesis

$P' \approx \varphi$ we conclude that $\varphi \xrightarrow{a} \varphi'$ and then $P \setminus \varphi \xrightarrow{a} Q \setminus \varphi'$. If a is not a synchronization action then $P \setminus \varphi \xrightarrow{a} Q \setminus \varphi$.

VI. Related Work

Several works have studied the correctness and conformance of composition of WS to security requirements.

A. Baouab et al. [7] show a run-time event-based approach to deal with the problem of monitoring conformance of interaction sequences. When a violation is detected, the program shows errors in dashboards. So the program does not stop before the violation occurred. J. Simmonds et al. [8] formalize sequence diagrams to express WS conversations and security requirements and then translate them to nondeterministic finite automata and generate monitors from NFA. Their WS conversation is extracted from the definition of simple services and so they did not consider the great number of WS conversations that will be provided with the composition of WS. D. Dranidis et al. [9] introduced an approach to verify the conformance of a WS 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 WS 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 nonconformance faults and does not react dynamically against these errors. Furhtermore, L. Ardissono et al. [10] 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.

R. Gay et al. [11] 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. W. She et al. [13] have developped an innovative security-aware service composition protocol with composition-time information flow control, which can reduce the execution-time failure rate of the composed composite services due to information flow control violations. This approach only guarantees that there are no access control violations at execution time but do not guarantee that there are not access control violations at runtime. Jose A. Martìn et al. [14] developed a framework based on the partial model checking technique for statically verifying whether a composition of WS satisfies cryptographic properties such as secrecy and authenticity.

VII. Conclusion and Future Work

The goal of this research is to introduce an automated formal approach for enforcing dynamically security policies on a choreography of WS using the rewriting technique. We used a formal language to express conversations of different participants and to express also security requirements. Then, we have shown how to restrict the progression of participant's behavior in order to satisfy security policies. Future work is concentrated on the optimization of this approach by reducing the number of synchronization actions that have been added to processes.

References

[1] I. Corporation, “Business process execution language for web services bpel-4ws,” http://www.ibm.com/developerworks/library/ws-bpel/, 2002.

[2] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon, “Web services choreography description language version 1.0,” W3C Working Draft, December 2004.

[3] M. Carbone, K. Honda, and N. Yoshida, “Theoretical aspects of communication-centred programming,” Electr. Notes Theor. Comput. Sci., vol. 209, 2008., pp. 125–133.

[4] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro, “Towards a formal framework for choreography,” in WETICE, 2005, pp. 107–112.

[5] G. D´ıaz, J. J. Pardo, M.-E. Cambronero, V. Valero, and F. Cuartero, “Automatic translation of ws-cdl choreographies to timed automata,” in EPEW/WS-FM, pp. 230–242, 2005.

[6] M. Langar, M. Mejri, and K. Adi, “Formal enforcement of security policies on concurrent systems,” J. Symb. Comput., vol. 46, no. 9, pp. 997–1016, 2011.

[7] A. Baouab, O. Perrin, and C. Godart, “An optimized derivation of event queries to monitor choreography violations,” in ICSOC, 2012, pp. 222– 236.

[8] J. Simmonds et al., “Runtime monitoring of web service conversations,” IEEE T. Services Computing, vol. 2, no. 3, 2009, pp. 223–244.

[9] D. Dranidis, E. Ramollari, and D. Kourtesis, “Run-time verification of behavioural conformance for conversational web services,” in ECOWS, 2009, pp. 139–147.

[10] L. Ardissono, R. Furnari, A. Goy, G. Petrone, and M. Segnan, “Monitoring choreographed services,” in Innovations and Advanced Techniques in Computer and Information Sciences and Engineering, 2007, pp. 283– 288.

[11] R. Gay, H. Mantel, and B. Sprick, “Service automata,” in Formal Aspects in Security and Trust, 2011, pp. 148–163.

[12] H. Yang, X. Zhao, Z. Qiu, G. Pu, and S. Wang, “A formal model for web service choreography description language (WS-CDL),” in 2006 IEEE International Conference on Web Services (ICWS 2006), September 2006, 18-22. Chicago, Illinois, USA, 2006, pp. 893–894.

[13] W. She, I. Yen, B. M. Thuraisingham, and E. Bertino, “Security-aware service composition with fine-grained information flow control,” IEEE T. Services Computing, vol. 6, no. 3, pp. 330–343, 2013.

[14] J. A. Mart´ın, F. Martinelli, I. Matteucci, E. Pimentel, and M. Turuani, “On the synthesis of secure services composition,” in Engineering Secure Future Internet Services and Systems - Current Research, 2014, pp. 140– 159.

 ISBN: 978-1-61208-376-6