Orchestration for Secure Multi-party Communications in Web-Services

Najah Ben Said, Takoua Abdellatif, Marius Bozga, Saddek Ben Salem, Axel Legay

Orchestration for Secure Multi-party Communications in Web-Services[☆]

Najah Ben Said[a], Takoua Abdellatif[b], Marius Bozga[c,d], Saddek Ben Salem[c], Axel Legay[a]

[a] *INRIA Rennes Bretagne Atlantique, Rennes, France*
[b] *Tunisia Polytechnic School, University of Carthage, Tunis, Tunisia*
[c] *Univ. Grenoble Alpes, VERIMAG, F-38000 Grenoble, France*
[d] *CNRS, VERIMAG, F-38000 Grenoble, France*

Abstract

Multi-party interactions in Web Service composition are hardly managed where parallel communications, while end-to-end security is respected, is difficult to be designed and verified. In this paper, we present an approach to handle and secure multi-party interactions in Web Service (WS) composition. A key ingredient of this methodology is to present the system composition at an abstract level as a component-based model where we verify and configure it securely. Then, we transform the model to generate secure orchestrator components that handle multi-party interactions. Afterwards, we generate accordingly *BPEL* processes where the security constrains are enforced as security WS-policies in the *BPEL* description of services. The framework we present is robust since founded on formal proves of the security properties. We validate this approach by securing a social network application called *Whens_App*.

Key words: web service orchestration, *BPEL*, component-based systems, information flow security, non-interference, secure-by-construction, automated verification.

1. Introduction

With the expansion of Web Services (WS) [1] deployed on the enterprise servers, cloud infrastructures and mobile devices, Web Service composition is currently a widely used technique to build complex Internet and enterprise applications. Orchestration languages, like *WS-BPEL* [2], allow rapidly developing composed WS by defining a set of activities binding sophisticated services. However, multi-party interactions involving several participants are relatively

[☆]This article extends the article [14] published at the ISoLA 2016 conference.The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement ICT-318772 (D-MILS).

less explored in actual standards such as *WS-BPEL* [3] and WS-CDL [4], especially for e-Business and Business-to-nBusiness (B2B) application integration. In several cases, responses might be time-critical and all suppliers might be required to receive the request and respond within a specified duration. The preparation of requests, the sending of requests and the receipt of responses might need coordination to concurrently execute a specific atomic tasks .

Additionally, within such functionality requirements, security-related problems and difficulties are raised where advanced skills and tools are required to ensure critical information security between different parties. Indeed, ensuring the communication security between parties who do not trust each other and where they must exchange messages in a synchronized and simultaneous way, is a tedious and very difficult task. It is important to track information flow and prevent illicit accesses by unauthorized services and networks; this task can be challenging when the service is complex or when the composition is hierarchical (the service is composition of composed services and atomic services). WS-security standards [5, 6] provide information flow security solutions for point-to-point inter-service communications but fall short in ensuring end-to-end information flow security in composed services. Furthermore, the *BPEL* language does not state any rules on how to properly secure service compositions.

In many situations, security is reduced to access control to prevent sensitive information from being read or modified by unauthorized users. However, access control is insufficient to regulate the propagation of information once released for processing by a program especially with non-trivial interactions and computations. Thus access control offers no guarantees about whether an information is subsequently protected. Deciding how to set access control permissions in complex systems is a difficult problem in itself. Equally, using cryptographic primitives that provides strong confidentiality and integrity guarantees, is also less helpful to ensure that the system obeys an overall security policy.

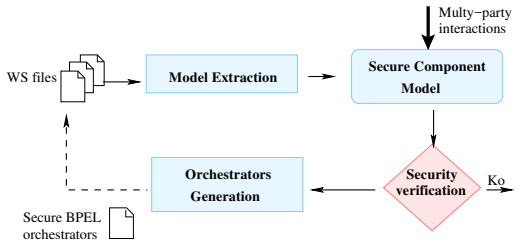

Figure 1: Secure work-flow for secure orchestrating multi-party interactions in WS composition.

In this paper, we propose a robust framework ensuring end-to-end information flow security while orchestrating WS. Figure 1 shows a work-flow overview of this framework. The service designer describes the behavior of his processes and defines security constraints. The constraints are expressed as authorization

rights, that is, a list of services owners and authorized readers for a subset of critical data. The WS processes can be expressed in different languages such as *BPEL* or Java (*BPEL* in our case) from which we extract their behavior into atomic components. Over these extracted components, we manually define and add a set of interactions considering user expectation.

The abstraction aim to verify an end-to-end information flow security on the system composition. Information flow security has been traditionally studied separately for language-based models [7, 8] (see also the survey [9]) and trace-based models [10, 11, 12, 13]. While the former mostly focus on verification of data-flow security properties in programming languages, the latter is treating security in event-based systems. In this work, we achieve a useful combination between both aspects, data-flow and event-flow security, in a single semantics model. In email exchange two types of information disclosure are possible: either by observing the email message content or by simply he observing email exchange occurrence, dates and frequency. we can collect information about a relation between two entities in a system by just observing events related to emails sending and receiving, while accessing the content of the emails would represent a collection of data exchanged between them. Therefore, we introduce and distinguish two types of non-interference, respectively *event non-interference* and *data non-interference*. For events, non-interference states that the observation of public events should not allow to deduce any information about the occurrence of secret events. For data, it states that there is no leakage of secret data into public ones.

Once our security properties are verified, we apply a correct-by-construction transformation that generates orchestrator components (services) that handle and manage the interactions execution in a decentralized way where the security constraints are preserved and based on which we generate an executable *BPEL* code. This approach is beneficial, as one can first ensure system requirements by dealing with a high-level formally specified model that abstracts implementation details and then derive a correct implementation through a series of transformations that terminates when an actual executable code is obtained.

This paper is an extension of [14] to automatically orchestrate Web services with secure multi-party interactions. To the best of our knowledge, there is no formal analysis on orchestrating and securing multi-party interactions in web services. Compared with the existing work, our contributions are the following:

- This method rely on a representation through an abstract high-level model allowing to run non-interference verification, then a transformation towards fully distributed model that is secure-by-construction. This approach is applied on composing WS and generating *BPEL* orchestrator's handling secure multi-party interactions.
- A distinction between two types of non-interference allowing to ensure security of both event and data while generating orchestrator components. In this approach, we ensure a security level-based isolation while handling interactions, and we exchange data from different security levels in a more permissive way between different orchestrator components.

- A tool-set implementation to generate *BPEL* orchestrator components and an evaluation of our approach through real case-study.

The paper is structured as follows. Section 3 presents the functional and security aspects of the adopted component-based framework. In Section 4, we present a correct-by-construction transformation approach to orchestrate WS compositions. Section 5 presents the implementation tool-set and we report experiments and discussion. Finally, Section 6 discusses the related work and Section 7 concludes and presents some lines for future work.

2. Running Example

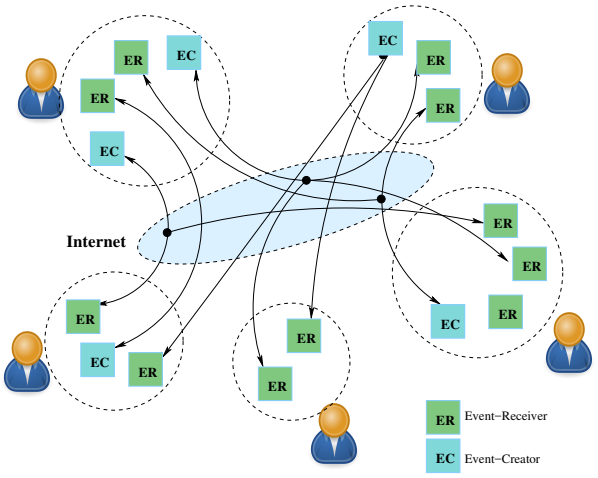

Figure 2: Overview of Whens_App application.

Throughout the paper, we consider a simplified social network application, called *Whens_App* that contains essentially two WS, Event-Receiver (ER) and Event-Creator (EC). The application is intended for organizing virtual multi-party events between ER and EC where participants can meet and exchange data as illustrated in Figure 2.

As social network application, *Whens_App* entails several security requirements. In this paper we focus on requirements related to information flow security: assuming that components are trustful and the network is insecure, *(1)* the interception and observation of exchanged data messages must not reveal any information about event organization and *(2)* confidentiality of classified data is always preserved and kept secret inter- and intra-components. We will show that both requirements are ensured by using security annotations for tracking events and data in the system. Then, we show how the annotated model can be automatically and systematically transformed towards a distributed implementation while preserving the security properties.

As previously mentioned, the input services can be defined in different programming languages (Java, WSDL, BPEL, ...), mainly, we simulate and present the behavior of the input processes in a component based models. In our particular case, we consider the BPEL language that we present in component model following transformation from [15].

3. Secure Component-Based Model

Systems are constructed from atomic components, that is, finite state automata or 1-safe Petri nets, extended with data and ports. Communication between components is achieved using multi-party interactions with data transfer.

Definition 1 (atomic component). *We present synchronous components and their semantics. The behavior of a synchronous component within a synchronous computation step is a 1-safe extended Petri net with given sets of initial and final places. When only final places are marked, termination may terminate by removing tokens from final places and putting tokens to initial places.*

An atomic component B is a tuple (L, X, P, T) where X is a set of variables and $N = (L, P, T)$ is an extended 1-safe Petri net. L is a finite set of places, P is a set of ports and $T \subseteq 2^L \times P \times 2^L$ is a finite set of transitions. A transition τ is a triple $({}^\bullet\tau, a, \tau^\bullet)$ where ${}^\bullet\tau$ is the input place of τ and the $\tau^\bullet$ is the output place of τ. A transition is labelled by a port p and (g_τ, f_τ) where g_τ is the guard of τ , that is a predicate on X and f_τ is the update function associated with τ, that is a state transformer defined on X.

Let $\mathcal{D}$ be a universal data domain, fixed. A valuation of a set of variables Y is a function $\mathbf{y} : Y \to \mathcal{D}$. We denote by $\mathbf{Y}$ the set of all valuations defined on Y. The semantics of an atomic component B is defined as the labelled transition system $\text{SEM}(B) = (Q_B, P_B, \underset{B}{\longrightarrow})$ where the set of states $Q_B = \mathcal{M} \times \mathbf{X}$ (where $\mathcal{M} = \{m : L \to \{0, 1\}\}$ is the set of 1-safe markings), the set of labels $\Sigma_B = P$ and transitions $\underset{B}{\longrightarrow}$ are defined by the rules:

$$\textsc{Atom}\ \frac{\tau = \ell \xrightarrow{p} \ell' \in T \quad \mathbf{x}''_p \in \mathbf{X}_p \quad g_\tau(\mathbf{x}) \quad \mathbf{x}' = f_\tau(\mathbf{x}[X_p \leftarrow \mathbf{x}''_p])}{(\ell, \mathbf{x}) \xrightarrow[B]{p(\mathbf{x}''_p)} (\ell', \mathbf{x}')}$$

That is, $(\ell', \mathbf{x}')$ is a successor of $(\ell, \mathbf{x})$ labelled by $p(\mathbf{x}''_p)$ iff (1) $\tau = \ell \xrightarrow{p} \ell'$ is a transition of T, (2) the guard g_τ holds on the current state valuation $\mathbf{x}$, (3) $\mathbf{x}''_p$ is a valuation of exported variables X_p and (4) $\mathbf{x}' = f_\tau(\mathbf{x}[X_p \leftarrow \mathbf{x}''_p])$ that is, the next-state valuation $\mathbf{x}'$ is obtained by applying f_τ on $\mathbf{x}$ previously modified according to $\mathbf{x}''_p$. Whenever a p-labelled successor exists in a state, we say that p is *enabled* in that state.

In our model, atomic components have exclusive access on their variables. Interactions between components take place only through explicit input/output

connectors. A connector defines a static communication channel from one output port p^{out} of a sender component B_i to a set of input ports $p_j^{\{in\}}$ in receiver components $\{B_j\}_{j=1..n}$ where $i \neq j$. The connector is denoted by the tuple $(p^{out}, p_i^{in}, \ldots, p_n^{in})$. Intuitively, when communication takes place, the value of var(pout) is assigned to var(pin).

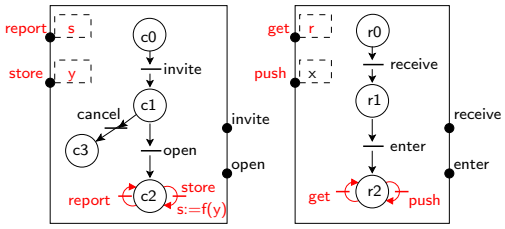

Figure 3: Example of atomic components

Figure 3 presents the atomic components used in the *Whens_App* application model. The Event Creator (left) coordinates an event lifetime (invite, open and cancel transitions), get raw information from participants (store) and delivers some information digests (report). The Event Receiver (right) enters an event (receive, enter), share information (push) and receive event digests (get). [Colors are explained later]

Composite components are obtained by composing atomic components $B_i = (L_i, X_i, P_i, T_i)_{i=1,n}$ through multi-party interactions. We consider that atomic components have pairwise disjoint sets of locations, ports, and variables i.e., for any two $i \neq j$ from $\{1..n\}$, we have $L_i \cap L_j = \emptyset$, $P_i \cap P_j = \emptyset$, and $X_i \cap X_j = \emptyset$.

A multi-party *interaction* a is a triple (P_a, G_a, F_a), where $P_a \subseteq \bigcup_{i=1}^{n} P_i$ is a set of ports, G_a is a guard, and F_a is a data transfer function. By definition, P_a uses at most one port of every component, that is, $|P_i \cap P_a| \leq 1$ for all $i \in \{1..n\}$. Therefore, we simply denote $P_a = \{p_i\}_{i \in I}$, where $I \subseteq \{1..n\}$ contains the indices of the components involved in a and for all $i \in I, p_i \in P_i$. G_a and F_a are both defined on the variables exported by the ports in P_a (i.e., $\bigcup_{p \in P_a} X_p$).

Let $\{B_i = (L_i, X_i, P_i, T_i)\}_{i=1,n}$ be a set of synchronous components defined on disjoint sets of variables and ports. Let γ be a set of interactions on ports $\cup_{i=1}^{n} P_i$ such that each interaction uses at most one port of every component, that is for all $a \in \gamma$, for all $i \in \{1, \ldots, n\}, |P_a \cap P_i| \leq 1$. The composition $\gamma(B_1, \ldots, B_n)$ is a partial operation defining the synchronous component $B = (X, P, N)$ where:

Definition 2 (composite component). *A composite component* $C = \gamma(B_1, \ldots, B_n)$ *is obtained by applying a set of interactions* γ *to a set of atomic components* $B_1, \ldots B_n$.

Let $B = \gamma(B_1, \ldots, B_n)$ be a composite component. Let $B_i = (L_i, P_i, T_i, X_i)$ and $\text{SEM}(B_i) = (Q_i, \Sigma_i, \xrightarrow[B_i]{})$ their semantics, for all $i = 1, n$. The semantics of C is the labelled transition system $\text{SEM}(C) = (Q_C, \Sigma_C, \xrightarrow[C]{})$ where the set of

states $Q_C = \otimes_{i=1}^{n} Q_i$, the set of labels $\Sigma_C = \gamma$ and the set of labelled transitions $\xrightarrow[C]{}$ is defined by the rule:

$$\textsc{Comp}\ \frac{\begin{array}{c} a = (\{p_i\}_{i\in I}, G_a, F_a) \in \gamma \qquad G_a(\{\mathbf{x}_{p_i}\}_{i\in I}) \qquad \{\mathbf{x}''_{p_i}\}_{i\in I} = F_a(\{\mathbf{x}_{p_i}\}_{i\in I}) \\ \forall i \in I.\ (\ell_i, \mathbf{x}_i) \xrightarrow[B_i]{p_i(\mathbf{x}''_{p_i})} (\ell'_i, \mathbf{x}'_i) \qquad \forall i \notin I.\ (\ell_i, \mathbf{x}_i) = (\ell'_i, \mathbf{x}'_i) \end{array}}{((\ell_1, \mathbf{x}_1), \ldots, (\ell_n, \mathbf{x}_n)) \xrightarrow[C]{a} ((\ell'_1, \mathbf{x}'_1), \ldots, (\ell'_n, \mathbf{x}'_n))}$$

For each $i \in I$, $\mathbf{x}_{p_i}$ above denotes the valuation $\mathbf{x}_i$ restricted to variables of X_{p_i}. The rule expresses that a composite component $C = \gamma(B_1, \ldots, B_n)$ can execute an interaction $a \in \gamma$ *enabled* in state $((\ell_1, \mathbf{x}_1), \ldots, (\ell_n, \mathbf{x}_n))$, iff (1) for each $p_i \in P_a$, the corresponding atomic component B_i can execute a transition labelled by p_i, and (2) the guard G_a of the interaction holds on the current valuation of variables exported on ports participating in a. Execution of interaction a triggers first the update function F_a which modifies variables exported by ports $p_i \in P_a$. The new values obtained, encoded in the valuation $\mathbf{x}''_{p_i}$, are then used by the components' transitions. The states of components that do not participate in the interaction remain unchanged.

Any finite sequences of interactions $w = a_1 ... a_k \in \gamma^*$ executable by the composite component starting at some given initial state q_0 is named a trace. The set of all traces w from state q_0 is denoted by $\textsc{traces}(C, q_0)$.

We call a trace any finite sequence of interactions $w = a_1 a_2 \cdots \in \gamma^*$ executable from a given initial state q_0. The set of all traces w from state q_0 is denoted by $\textsc{traces}(C, q_0)$.

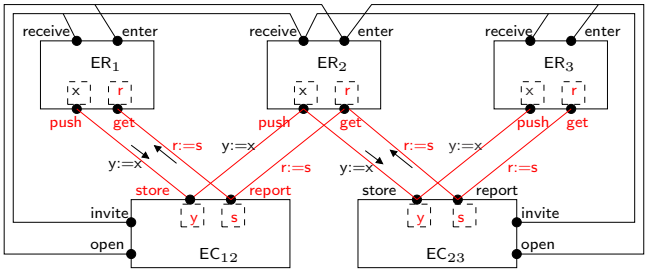

Figure 4: Example of composite component

Figure 4 presents a simplified composite component for an instance of the *Whens_App* application with two event creators and three event receivers. Interactions are represented using connecting lines between the interacting ports. Binary interactions (push store) and (report get) include data transfers between components, that is, assignments of data across interacting components.

3.1. Information Flow Security

We consider transitive information flow policies expressed on system variables and we focus on the non-interference properties. We restrict ourselves to

confidentiality and we ensure that no illegal flow of information exists between variables having incompatible security levels.

Formally, we represent security domains as finite lattices $\langle \mathbb{S}, \sqsubseteq \rangle$ where $\mathbb{S}$ denotes the security levels and $\sqsubseteq$ the *flows to* relation. For example, a security domain with two levels *High* (H), *Low* (L) and where information is allowed to flow from *Low* to *High* is $\langle \{L, H\}, \{(L, L), (L, H), (H, H)\} \rangle$. For clarity sake of the paper, we will use H and L levels to present our approach to handle security.

Let $C = \gamma(B_1, \ldots B_n)$ be a composite component, fixed. Let X (resp. P) be the set of all variables (resp. ports) defined in all atomic components $(B_i)_{i=1,n}$. Let $\langle \mathbb{S}, \sqsubseteq \rangle$ be a security domain, fixed.

Definition 3 (security assignment σ). *A security assignment for component C is a mapping $\sigma : X \cup P \cup \gamma \to \mathbb{S}$ that associates security levels to variables, ports and interactions such that, moreover, the levels of ports and interactions match, that is, for all $a \in \gamma$ and for all $p \in P$ it holds $\sigma(p) = \sigma(a)$.*

The security levels for ports and variables track the flow of information along computation steps within atomic components. The security levels for interactions track the flow of information along inter-component communication. We consider that deducing event-related information represent a risk that should be handled while controlling the system's information flow in addition to data flows. End-to-end security is defined according to transitive non-interference.

Let σ be a security assignment for C, fixed. For a security level $s \in \mathbb{S}$, we define $\gamma \downarrow_s^\sigma$ the restriction of γ to interactions with security level at most s that is formally, $\gamma \downarrow_s^\sigma = \{a \in \gamma \mid \sigma(a) \sqsubseteq s\}$. For a security level $s \in \mathbb{S}$, we define $w|_s^\sigma$ the projection of a trace $w \in \gamma^*$ to interactions with security level lower or equal to s. Formally, the projection is recursively defined on traces as $\epsilon|_s^\sigma = \epsilon$, $(aw)|_s^\sigma = a(w|_s^\sigma)$ if $\sigma(a) \sqsubseteq s$ and $(aw)|_s^\sigma = w|_s^\sigma$ if $\sigma(a) \not\sqsubseteq s$. The projection operator $|_s^\sigma$ is naturally lifted to sets of traces W by taking $W|_s^\sigma = \{w|_s^\sigma \mid w \in W\}$.

For a security level $s \in S$, we define the equivalence $\approx_s^\sigma$ on states of C. Two states q_1, q_2 are equivalent, denoted by $q_1 \approx_s^\sigma q_2$ iff (1) they coincide on variables having security levels at most s and (2) they coincide on control states having outgoing transitions labeled with ports with security level at most s. We are now ready to define the two types of non-interference respectively *event non-interference (ENI)* and *data non-interference (DNI)*.

Definition 4 (event/data non-interference). *The security assignment σ ensures event (ENI) and data non-interference (DNI) of $\gamma(B_1, \ldots, B_n)$ at security level s iff,*

$$(ENI) \quad \forall q_0 \in Q_C^0 : \textsc{traces}(\gamma(B_1, \ldots, B_n), q_0)|_s^\sigma = \textsc{traces}((\gamma \downarrow_s^\sigma)(B_1, \ldots, B_n), q_0)$$

$$(DNI) \quad \forall q_1, q_2 \in Q_C^0 : \; q_1 \approx_s^\sigma q_2 \Rightarrow \forall w_1 \in \textsc{traces}(C, q_1), w_2 \in \textsc{traces}(C, q_2) : \; w_1|_s^\sigma = w_2|_s^\sigma \Rightarrow \forall q_1', q_2' \in Q_C : \; q_1 \xrightarrow[C]{w_1} q_1' \wedge q_2 \xrightarrow[C]{w_2} q_2' \Rightarrow q_1' \approx_s^\sigma q_2'$$

Moreover, σ is said secure *for a component $\gamma(B_1, \ldots, B_n)$ iff it ensures both event and data non-interference, at all security levels $s \in \mathbb{S}$.*

Both variants of non-interference express some form of indistinguishability between several states and traces of the system. For instance, an attacker that can observe the system's variables and occurrences of interactions at security level s_1 must not be able to distinguish neither changes on variables or occurrence of interactions having higher or incomparable security level s_2.

The running example presented in Figures 3 and 4 is annotated with two levels of security *Low* (in black) and *High* (in red). With this assignment, the exchange of information during the event and some related data are *High* whereas the event initiation is *Low*.

3.2. Noninterference Checking

In our previous work [16], we established sufficient syntactic conditions that reduce the verification of non-interference to local constrains checking on transitions (intra-component) and interactions (inter-components). We recall these conditions hereafter as they are going to be used later in section 4 for establishing security correctness of the decentralized component model. Indeed, these conditions offer a syntactic way to ensure both event and data non-interference and therefore to obtain preservation proofs for along decentralization.

Definition 5 (security conditions). *Let $C = \gamma(B_1, \ldots, B_n)$ be a composite component and let σ be a security assignment. We say that C satisfies the security conditions for security assignment σ iff:*

(i) the security assignment of ports, in every atomic component B_i is locally consistent, that is, for every pair of causal transitions:
$\forall \tau_1, \tau_2 \in T_i \ : \ \tau_1 = \ell_1 \xrightarrow{p_1} \ell_2, \ \tau_2 = \ell_2 \xrightarrow{p_2} \ell_3 \Rightarrow (\ell_1 \neq \ell_2 \Rightarrow \sigma(p_1) \sqsubseteq \sigma(p_2))$
and for every pair of conflicting transitions:
$\forall \tau_1, \tau_2 \in T_i \ : \ \tau_1 = \ell_1 \xrightarrow{p_1} \ell_2, \ \tau_2 = \ell_1 \xrightarrow{p_2} \ell_3 \Rightarrow \sigma(p_1) = \sigma(p_2)$

(ii) all assignments $x := e$ occurring in transitions within atomic components and interactions are sequential consistent, in the classical sense:
$\forall y \in use(e) \ : \ \sigma(y) \sqsubseteq \sigma(x)$

(iii) variables are consistently used and assigned in transitions and interactions:
$\forall \tau \in \cup_{i=1}^{n} T_i, \ \forall x, y \in X \ : \ x \in def(f_\tau), y \in use(g_\tau) \Rightarrow \ \sigma(y) \sqsubseteq \sigma(p_\tau) \sqsubseteq \sigma(x)$
$\forall a \in \gamma, \ \forall x, y \in X \ : \ x \in def(F_a), y \in use(G_a) \Rightarrow \sigma(y) \sqsubseteq \sigma(a) \sqsubseteq \sigma(x)$

(iv) all atomic components B_i are port deterministic:
$\forall \tau_1, \tau_2 \in T_i \ : \ \tau_1 = \ell_1 \xrightarrow{p} \ell_2, \ \tau_2 = \ell_1 \xrightarrow{p} \ell_3 \Rightarrow (g_{\tau_1} \wedge g_{\tau_2})$ *is unsatisfiable*

The first family of conditions (i) is similar to Accorsi's conditions [17] for excluding causal and conflicting places for Petri net transitions having different security levels. Similar conditions have been considered in [18, 19] and lead to

more specific definitions of non-interferences and bi-simulations on annotated Petri nets. The second condition (*ii*) represents the classical condition needed to avoid information leakage in sequential assignments. The third condition (*iii*) tackles covert channels issues. Indeed, (*iii*) enforces the security levels of the data flows which have to be consistent with security levels of the ports or interactions (e.g., no low level data has to be updated on a high level port or interaction). Such that, observations of public data would not reveal any secret information. Finally, condition (*iv*) enforces deterministic behavior on atomic components.

The following result, proven in [16], states that the security conditions are sufficient to ensure both event and data non-interference.

Theorem 1. *Whenever the security conditions hold, the security assignment σ is secure for the composite component C.*

For example, the security conditions hold for the security assignment considered for the running example in Figures 3 and 4. Notice that local consistency is ensured in both atomic components: the security level can only increase from *Low* to *High* along causal transitions and no choices exist between *Low* and *High* transitions. Equally, notice that no *High* data is assigned on *Low* interactions.

4. Automated Generation of Secure Orchestrators

In this section, we describe an automated way to generate orchestrator components that enforces the information flow security in the *BPEL* processes compositions while handling the interaction executions.

This generation introduces (1) a transformation on atomic components behavior where we introduce Send/Receive message passing and (2) using adaptors to handle the execution of interactions. Whenever a component needs to interact, it publishes an offer, that is the list of its enabled ports, then wait for a notification from the orchestrator indicating which interaction has been chosen, and then resume its execution. From his side, every orchestrator component handles a subset of interactions, that is, checks them for enabledness and schedule them for execution accordingly. The interface between components and orchestrator provides ports for receiving offers and notifying the ports selected for execution.

Using this decentralization approach, the preservation of information flow security in the system composition is achieved by imposing few restrictions on the structure of the orchestrator components and providing additional communication ports at atomic component level to exchange messages and data security between them. We show that the security assignment from the original model is naturally lifted to the decentralized model and consequently, non-interference is preserved along the transformation.

Let $C = \gamma(B_1, \cdots, B_n)$ be a composite component and σ be a secure assignment for C which satisfies the security conditions for non-interference.

4.1. Atomic Components

The transformation of atomic components consists in breaking atomicity of transitions. Precisely, each transition is split into two consecutive steps: (1) an offer that publishes the current state of the component, and (2) a notification that triggers an update function and resume local computation. The intuition behind this transformation is that the offer transition correspond to sending information about component's intention to interact with the orchestrator component and the notification transition corresponds to receiving the answer from the orchestrator, once an interaction has been completed. Update functions can be then executed concurrently and independently by components upon notification reception.

To protect information flow, distinct offer ports o_s and interaction counters n_s are introduced for every security level defined initially at centralized atomic components. Thus, offers and corresponding notifications have the same security level, and moreover, no information about execution of interactions is revealed through the observation of interaction counters.

Definition 6 (transformed atomic component). *Let $B = (L, X, P, T)$ be an atomic component within C. The corresponding transformed S/R component is $B^{SR} = (L^{SR}, X^{SR}, P^{SR}, T^{SR})$:*

- $L^{SR} = L \cup L^{\perp}$, *where* $L^{\perp} = \{\perp_\ell \mid \ell \in L\}$
- $X^{SR} = X \cup \{e_p\}_{p \in P} \cup \{n_s | s \in \mathbb{S}\}$ *where e_p is a fresh boolean variable indicating whether port p is enabled, and n_s is a fresh integer variable called interaction counter for security level s.*
- $P^{SR} = P \cup \{o_s \mid s \in S\}$. *The offer ports o_s export the variables* $X_{o_s} = \{n_s\} \bigcup \{\{e_p\} \cup X_p \mid \sigma(p) = s\}$ *that is the interaction counter n_s, the newly added variable e_p and the variables X_p associated to ports p with security level s. For other ports, the set of variables exported remains unchanged.*
- *For each state $\ell \in L$, let S_ℓ be the set of security levels assigned to ports labeling all outgoing transitions of ℓ. For each security level $s \in S_\ell$, we include the offer transition* $\tau_{o_s} = (\perp_\ell \xrightarrow{o_s} \ell) \in T^{SR}$, *where the guard g_{o_s} is true and f_{o_s} resets variables e_p to false, for all ports p with security level s.*
- *For each transition* $\tau = \ell \xrightarrow{p} \ell' \in T$ *we include a notification transition* $\tau_p = (\ell \xrightarrow{p} \perp_{\ell'})$ *where the guard g_p is true and the function f_p applies the original update function f_τ on X, sets e_r variables to g_{τ_r} for every port $r \in P$ such that* $\tau_r = \ell' \xrightarrow{r} \ell'' \in T$ *and increments n_s.*

We introduce now the extended security assignment for transformed atomic components B^{SR}. Intuitively, all existing variables and ports from B keep their original security level, whereas the newly introduced ones are assigned such that to preserve the security conditions of the transformed component.

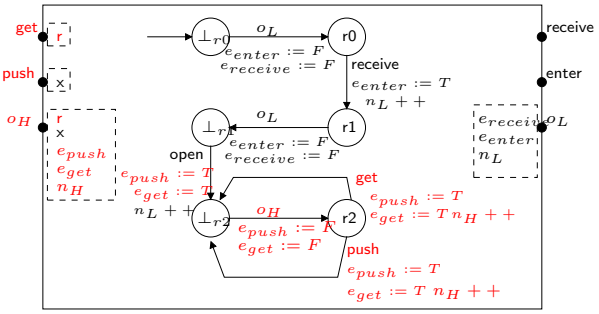

Figure 5: Transformation of atomic components illustrated on the Event Receiver

Definition 7 (security assignment σ^{SR} for B^{SR}). *The security assignment σ^{SR} is the extension of the original security assignment σ to variables X^{SR} and ports P^{SR} from B^{SR} as follows:*

$$\sigma^{SR}(x) = \begin{cases} \sigma(p) & \text{if } x = e_p \text{ and } p \in P \\ s & \text{if } x = n_s \text{ and } s \in S \\ \sigma(x) & \text{otherwise, for } x \in X^{SR} \end{cases} \qquad \sigma^{SR}(p) = \begin{cases} s & \text{if } p = o_s \text{ and } s \in \mathbb{S} \\ \sigma(p) & \text{otherwise, for } p \in P^{SR} \end{cases}$$

As example, the component transformation and the extended security assignment for the Event Receiver are depicted in Figure 5. Variables $n_L, e_{invite}, e_{open}$ and the offer port o_L are assigned to *Low*. Variables n_H, e_{push}, e_{get} and the port o_H are assigned to *High*. Ones can check that this assignment obeys all the (local) security conditions related to B^{SR}.

Actually, security conditions are preserved along the proposed transformation of atomic components with respect to extended security assignment. The following lemma formalizes this result.

Lemma 1. *B^{SR} satisfies the security conditions with security assignment σ^{SR}.*

Proof 1. *easy check, security conditions hold by definition of B^{SR} and σ^{SR}.*

4.2. Secure Orchestrator Generation

Orchestrator consists of a set of components, each in charge of the execution of a subset of interactions from the original component model. Every such orchestrator component is a controller that, iteratively, receives offers from the transformed atomic components, computes enabled interactions and schedule them for execution.

In this paper, we consider orchestrator components handling a conflict-free partitioning of interactions, as in [20]. Two interactions a_1 and a_2 are in conflict iff either (i) they share a common port p (i.e $p \in a_1 \cap a_2$) or (ii) there exist two conflicting transitions at a local state ℓ of a component B_i that are labeled with ports p_1 and p_2, where $p_1 \in a_1$ and $p_2 \in a_2$. Conflict-free partitioning allows orchestrator to run fully independently of each other, that is, local decisions

taken on every orchestrator component about executing one of its interactions do not interfere with others.

Moreover, in order to ensure information flow security, we impose an additional restriction on interaction partitioning, that is, the subset of interactions handled within every orchestrator component must have the same security level. Intuitively, this restriction allows us to enforce by construction the security conditions for all orchestrator and later, for the system composition.

Bearing this in mind, let us observe that if the original system satisfies the security conditions then the partitioning of interactions according to their security level is conflict-free. That is, no conflict exists between interactions with different security levels - this simply follows from the condition (i) on the labeling of conflicting transitions. Therefore, for the sake of simplicity of presentation, we restrict hereafter our construction to the partitioning according to security levels. For every security level s we consider one orchestrator component, $(Orch_s)$, handling the subset of interactions $\gamma_s = \{a \in \gamma \mid \sigma(a) = s\}$ with security level s.

Definition 8 (Orchestrator component at level s ($Orch_s$)). *The component $Orch_s = (L^O, X^O, P^O, T^O)$ handling γ_s is defined as:*

- *Set of places L^O is the union of waiting places $\{\{w_i\} \mid i \in \{1, \ldots, n\}\}$ and receive places $\{\{r_i\} \mid i \in \{1, \ldots, n\}\}$ for all $B_i \in participants(\gamma_s)$ and sending places $\{s_p \mid p \in ports(\gamma_s)\}$.*
- *Set of variables X^O is the union of notification variables $\{\{n_{is} \mid i \in \{1, \ldots, n\}\}$ for every security level s defined in every component $B_i \in participants(\gamma_s)\}\}$ and the variables by the offer port $\{\{x_p\} \cup X_p \mid p \in ports(\gamma_s)\}$*
- *Set of ports $P^O = \{\{o_{si}\} \mid i = \{1, \ldots, n\}\} \cup \{p \mid p \in ports(\gamma_s)\}$ where offer ports o_{is} are associated to variables n_{is}, x_p, and X_p from all component $B_i \in participants(\gamma_s)$ and ports p are associated to variables X_p.*
- *Set of transitions $T^O \subseteq 2^{L^O} \times P^O \times 2^{L^O}$. A transition τ is a triple $({}^\bullet\tau, p, \tau^\bullet)$, where ${}^\bullet\tau$ is the set of input places of τ and $\tau^\bullet$ is the set of output places of τ. We introduce three types of transitions:*
  - *receiving offers (w_i, o_{si}, r_i) for all components $B_i \in participants(\gamma_s)$.*
  - *executing interaction $(\{r_i\}_{i \in I_2}, a, \{s_{pi}\}_{i \in I_2})$ for each interaction $a \in \gamma_s$ such that $a = \{p_i\}_{i \in I_2}$, where I_2 is the set of components involved in a. To this transition we associate the guard $[G_a \wedge \bigwedge_{p \in a} x_p]$ and we apply the original update function F_a on $\cup_{p \in a} X_p$.*
  - *sending notification (s_p, p, w_i) for all ports p and component $B_i \in participants(\gamma_s)$.*

Definition 9 (security assignment σ^{SR} for $Orch_s$). *The security assignment σ^{SR} is built from the original security assignment σ. For variables X^{Orch} and*

ports P^{Orch} *of the* $Orch_s$ *component that handles* γ_s*, we define*

$$\sigma^{SR}(x) = \begin{cases} \sigma(x) & \text{if } x \in X_p \text{ and } s \sqsubseteq \sigma(x) \\ s & \text{otherwise} \end{cases} \qquad \sigma^{SR}(p) = s \text{ if } p \in P^O$$

The above definition enforces the security conditions for $Orch_s$ adaptors.

Lemma 2. *$Orch_s$ satisfies the security conditions with security assignment σ^{SR}.*

Proof 2. *Trivial check for conditions (i, iv). The condition (ii) on sequential consistency is also valid, even if some (replicated) variables within $Orch_s$ are upgraded to level s. On one hand, these variables, if any, were exclusively* used *(e.g., within guards, or left-hand sides of assignments) and never* defined *in interactions from γ_s. On the other hand, all defined variables have the security level greater than s. Same reasoning applies for the condition (iii) with respect to ports.*

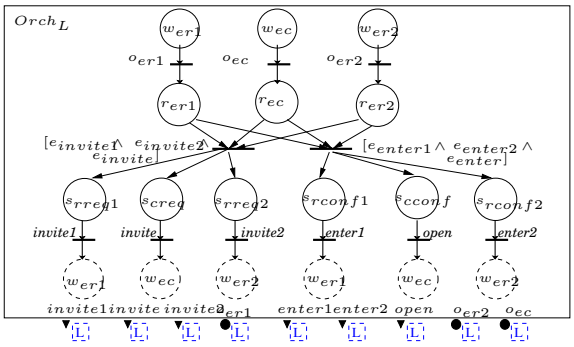

Figure 6: Generated orchestrator component $Orch_L$ for a simplified *Whens_App* system composed of two Event_Receivers and one Event_Creator

The extended security assignment σ^{SR} for $Orch_s$ variables and ports is defined as follows. All ports are annotated with security level s. Regarding variables, σ^{SR} maintains the same security level for all variables having their level greater than s in the original model and *upgrades* the others to s. That is, all variables within the $Orch_s$ component will have security level at least s. This change is mandatory to ensure consistent transfer of data in offers (resp. notifications) between atomic components and $Orch_s$.

4.3. System Composition

As a final step, the decentralized model C^{SR} is obtained as the composition $\gamma^{SR}(B_1^{SR}, ..., B_n^{SR}, (Orch_s)_{s \in \mathbb{S}})$ involving the transformed components B_i^{SR} and components $Orch_s$. The set γ^{SR} contains S/R interactions and is defined as follows:

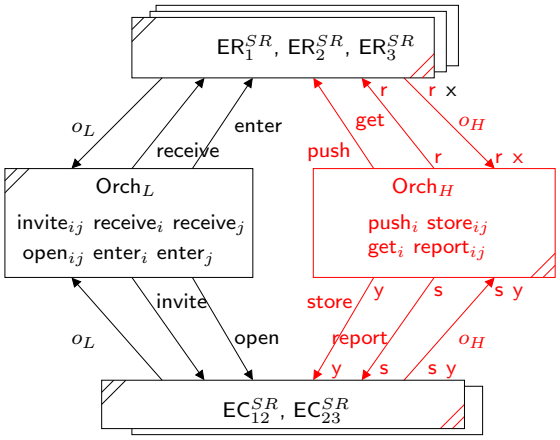

Figure 7: Decentralized model for the *Whens_App* example

- for every component B_i^{SR} participating in interactions having security level s, include in γ^{SR} the *offer* interaction $(B_i^{SR}.o_s, Orch_s.o_i)$ associated with the transfer of data from the component port o_s to the $Orch_s$ component port o_i.
- for every port p in component B_i^{SR} with security level s, include in γ^{SR} the *notification* interaction $(Orch_s.p, B_i.p)$ associated with the transfer of the subset of X_p variables having security level at least s from the $Orch_s$ component port p to the component port p. Actually, these are the only variables that could have been modified by an interaction having level s.

The security assignment σ^{SR} is naturally lifted from offer/notification ports to the interactions of γ^{SR}. Intuitively, every S/R interaction involving component $Orch_s$ has security level s. The construction is illustrated for the running example in Figure 7. We omitted the representation of ports and depict only the interactions and their associated data flow. In particular, consider the x variable of Event Receiver which is upgraded to H when sent to $Orch_H$ and not sent back on the notification of the push interaction.

The following theorem states our main result, that is, the constructed two-layer S/R model satisfies the security conditions by construction.

Theorem 2. *The decentralized component* $C^{SR} = \gamma^{SR}(B_1^{SR}, ..., B_n^{SR}, (Orch_s)_{s \in S})$ *satisfies security conditions for the security assignment* σ^{SR}.

Proof 3. *From lemma 1 and 2 all security conditions related to transformed components and orchestrator components are satisfied. The only remaining condition (iii) concerns the assignment of data along S/R interactions. As all the variables in* $Orch_s$ *have been eventually upgraded to level* s*, the assignment within offer interactions is consistent. Similar for notifications at level* s*, their*

assignment is restricted by construction to variables having security level at least s.

Example 1. *Figure 8 (a) presents a data transfer between Event_Creator and an Event_Receiver on the synchronized interaction get_rep where the variable r from component Event_Receiver is assigned to the variable s in component Event_Creator if the variable active is true. The variable s, r are tagged with H annotation and the variable active is tagged with L annotation. In the decentralized model shown in Figure 8 (b), the $Orch_H$ component executes interaction get. To this end, variables s, r are imported into the same security level of variables s′, r′, while the variable active is imported into a higher security level variable active′, through the corresponding offer port. Once the interaction takes place, s′ is copied back to s on the notification transition. No copy is performed back to the r and active variables. Here we manage different level variable in the same Orch scheduler.*

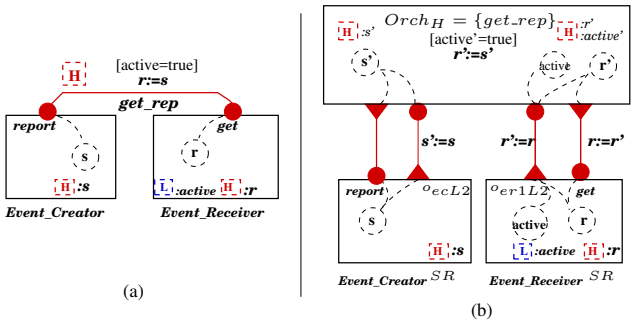

Figure 8: Secure Data Exchange between atomic and orchestrator components.

5. Implementation

In this section, we illustrate a complete design flow for generating secure distributed code represented in Figure 9. The implementation is based on the use of *secureBIP* framework as a platform. The white strong lined boxes represent modules that we implemented while the shaded strong lined ones represent modules that already exists and we modified to encompass security. Based on *secureBIP* framework [21], we implement these modules in Java language and we generate *BPEL* processes for the system composition. In this architecture, the flow consists on configuring security at two levels, first at the abstract model and second depending on target platform. Hereafter, we first present the transformation of orchestrator components to *BPEL* processes and we introduce the implement annotation model, then we present the *secureBIP* tool-set for different implementation steps and design choices.

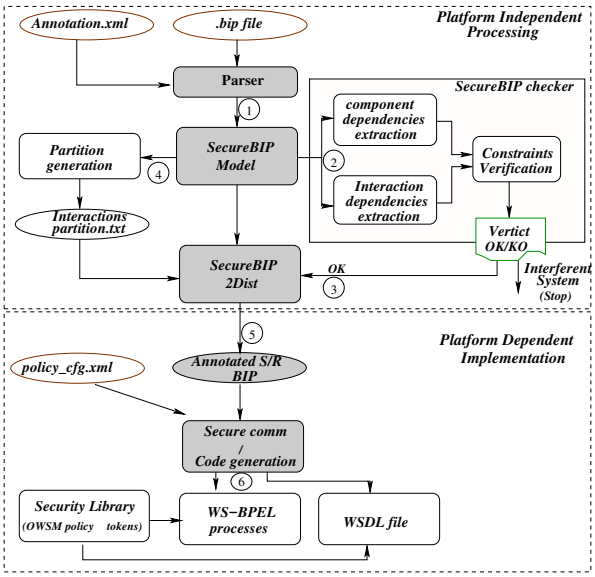

Figure 9: Tool-set Architecture Overview.

5.1. BPEL Process for Orchestrator Components

BPEL provides structuring mechanisms several WS into a new one. We particularly focus on *WS-BPEL* processes which compose services from activities, that are either (1) *basic* such as receive, reply, invoke, assign, throw, exist, or (2) *structured* such as sequence, if, while, repeatuntil, pick, flow.

Here the transformation consists mainly on generating *BPEL* code for orchestrator component to manage the execution of system composition. Many transformations from Petri-nets into *BPEL* has been given in the literature. Our work is based on a formally verified transformation presented in detailed mapping between WF-nets to *BPEL*. For more details, we refer to a technical report [22]. The transformation in the reverse direction, from *BPEL* to component-based model is given in our previous work [15]. This transformation exploits the behavior of the orchestrator component and defines the different sequences and flows in each behavior. Clearly, the sequence allows for the most straightforward mapping onto *BPEL*. From any source place to any sink place, that are connected by different transitions each representing an activity in the atomic component, we consider it as a switch that can be either explicit (where condition are defined over expressions) or implicit (only with messages). A loop on a place is transformed into a while construct.

More particularly, the behavior of an orchestrator component is generic, presented as a set of $\langle flow.../\rangle$ depending on the number of handled interactions in each component. An interaction in an *Orch* component can only take place if all offers from the involved components are received. Such structure

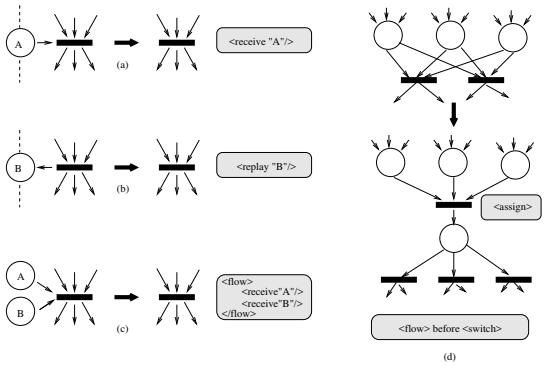

Figure 10: The representation of main transitions of the orchestrator component in *BPEL* language.

is transformed in a *BPEL* process as ⟨*flow.../*⟩ construct. As presented in Figure 10, (a) the offer transitions are transformed into ⟨*receive.../*⟩ activity and (b)the notify transitions are translated into ⟨*replay.../*⟩ activities in the flow. The transition representing interaction execution are transitions with multiple input places that are translated to an internal assign activity that executes the corresponding assignment on an interaction with a condition set to "All" (Figure 10 (c) and (d)). The join condition states that all interacting component should be ready to execute. Considering the example given earlier (Figure 6) of generating an orchestrator $Orch_L$ to handle interactions from the same security level L between two *Event_Receivers* and one *Event_Creator*. Figure 11 is a representation of the $Orch_L$ component into a *BPEL* orchestrator process. This process is basically a flow over a set of received offer messages that contains a set of updated variables. According to the condition expression value, one of the interactions is executed.

5.2. Security Labels Implementation

To track information flow in the system, we implemented the Decentralized Label Model [23]. This model provides a universal labelling scheme where security labels (or levels) are expressed using set of policies. A confidentiality label L contains (1) an owner set, denoted *O(L)*, that are principals representing the originating sources of the information, and (2) contains for each owner $o \in O(L)$ a set of readers, denoted *R(L,o)*, representing principals to whom the owner o is willing to release the information. The association of an owner o and a set of readers $R(o)$ defines a policy. Principals are ordered using an *acts_for* partial order relation (denoted $\prec$) which is a delegation mechanism that enables a principal to pass his rights to another principal (e.g., $o_1 \prec o_2$ states that o_2 can act for o_1). A security domain is defined over the set of confidentiality labels by

```
<flow name="InteractionsExecutionOrch_L">
 <receive partner="EventCreator" variable="e_inviteECRequest"
  operation="sendOfferECRequest" container="OER"/>
  ...
 <switch name="switchCCTypeFlag">
  <case> <condition>
   $e_inviteEventCreatorRequest.payload = 'true' &...
   </condition>
   <assign name="assignEventCreatorRequest"><copy>
    <from expression="$e_inviteECRequest.payload= 'false'"/>
    <to variable="p_inviteECReplay" part="payload"/>
   </copy></assign>
    ...
   <flow>
    <reply partner="EC" operation="ReplayEventCreation"
              variable="p\_enter1ECReplay"/>
    ...
   </flow>
  </case>
  <otherwise>
   <condition>$e_enterECRequest.payload= 'true'
   & ...</condition>
    <assign name="assignECRequest">
     <copy><from expression="$e_enterECRequest.payload = 'false'"/>
      <to variable="p_enterECReplay" part="payload"/>
    </copy></assign>
    ...
    <flow>
         <reply partner="EC" operation="ReplayEC"
                    variable="p_enter1ECReplay"/>
    ...
    </flow>
   </otherwise>
  </switch>
</flow>
```

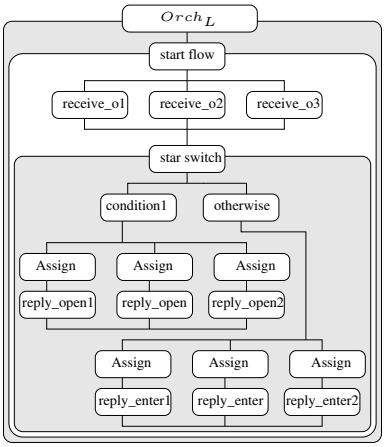

Figure 11: BPEL process generation of an orchestrator component.

using a *flows to* relation defined as follows:

$$L_1 \sqsubseteq L_2 \quad \equiv \quad \forall o_1 \in O(L_1).\ \forall o_2 \in O(L_2).\ o_1 \prec o_2 \wedge \\ \forall r_1 \in R(L_1, o_1).\ \exists r_2 \in R(L_2, o_2).\ r_1 \prec r_2$$

The intuition behind the *flows to* relation $\sqsubseteq$ above is that *(1)* the information can only flow from one owner o_1 to either the same or a more powerful owner o_2 where o_2 can act for o_1 and *(2)* the readers allowed by $R(L_2, o)$ must be a subset of the readers allowed by $R(L_1, o)$ where we consider that the readers allowed by a policy include not only the principals explicitly mentioned but also the principals able to act for them.

In our setting for *BPEL* WS, the principals used to define the *acts_for* relation and the security domain are obtained from *BPEL* partner-links that correspond to WS URI. That is, principals can be either *BPEL* processes or atomic WS in some primitive language. The designer expresses his security policy by tagging *BPEL* variables in each process using *DLM* labels. The security domain and these annotations are then transposed as such on atomic components.

5.3. Abstract Model Configuration

Additionally to the system functional model, security annotation is provided in a configuration file (Annotations.xml) that contains the *acts_for* relations and labels to different ports and data in each atomic component. Figure 12 present the configuration file for the *Whens_App* abstract model. We extend the system

model parser to extract labels from Annotations.xml file and we associate them to their corresponding ports and data types in the *secureBIP* model. Next, the *secureBIP checker* tool browses all atomic components and interactions in the model to extract events dependencies at each local state (incoming and outgoing port labelled transitions) and data dependencies at different transition's and interaction's actions and checks their label consistency. In the case where tool verdict is positive, the tool generates automatically an interaction partition file that describes the set of interactions that each orchestrator component would manage. This file is used as input by *secureBIP2Dist* to generate an annotated S/R model. The *secureBIP2Dist* generator is modified to encompass modifications in decentralized model as well as rules for annotations propagation.

```
<config>
   <acts_for><authority> EC:ER1,ER2; </authority></acts_for>
   <var_config>
        <variable name="s" component="EC" label="EC:ER1,ER2">
        <variable name="r1" component="ER1" label="EC:ER1,ER2">
        <variable name="r2" component="ER2" label="EC:ER1,ER2">
          ...
   </var_config>
   <port_config>
        <port name="invite" component="EC" label="_:_">
        <port name="open" component="ER1" label="_:_">
        <port name="store" component="EC" label="EC:ER1,ER2">
          ...
   </port_config>
</config>
```

Figure 12: Platform independent configuration

5.4. Platform-Dependent Configuration

Here the system designer provides configuration file that maps security-policies to be used to ensure confidentiality and integrity for data and ports to secure interactions between atomic S/R and orchestrator components. To preserve confidentiality, we use encryption and for integrity we use digital signature. We assume that the generated code is running on trusted hosts where it is safe to generate and store encryption keys. *Security Library* contains different tokens for encryption protocols and functions that, following the policy-configuration file (policy-cfg.xml), the code generator selects messages to secure at communications.

```
<platform_config>
   <security level="EC:ER1,ER2"><encryption name="rsa_encrypt_2048"/>
        <signature token="X.509"/></security>
   <security level="_:_"><encryption name=""/>
        <signature token=""/></security>
</platform_config>
```

Figure 13: Platform dependent configuration

Each component in the abstract model corresponds to a *BPEL* process that interprets its behavior. The configuration states the encryption and signature mechanisms for each defined security level, that is, for variables and ports that need to be secured following the secure abstract annotations. When only the

variable in the process is configured to be confidential and it is transiting in a low level security channel, encryption and signature is only applied to the value of the variable before sending it (which corresponds to the payload of the message). However, when the interaction (channel) connecting two high level security ports, the hole connection is considered to be confidential and a session should be created where we enforce message privacy (not only the payload of a message should be encrypted and even header where the sender and receiver of messages are hidden).

To ensure this, we mainly relied on the use of Oracle Web Service Manager (OWSM) to ensure the security of generated *BPEL* orchestrator. Encryption/decryption mechanisms (function, libraries and services) are implemented using *WS-security* tokens. A selected security mechanism is invoked if the variable that we intend to send belongs to a specific sensitive domain. The payload of a message is encrypted and then sent back to the destination service. For integrity sake, security tokens signing messages are added as ws-security policies in the WSDL file describing the service and are checked at each message send and receive. Authentication mechanisms and session encryption are also introduced in WSDL file when the used ports are configured to be secured.

5.5. Discussion and Experiments

Here we introduce configuration according to the propagated annotation in the distributed model using the configuration file where we specify authentication and encryption mechanisms. The executions is performed on an Intel Code 2Duo 2GHz with 4GB RAM memory running Linux Ubuntu. For generation of the certificates for signature and encryption, we use X.509 WS-security tokens with an asymmetric encryption algorithm (RSA) with 2048bit key size.

As an Evaluation of our approach performance, we consider three execution scenarios:

- P1: we consider a set of 100 S/R components within two orchestrator components, communicating between each other with a defined set of interactions.
- P2: we consider set of 10 S/R components within two orchestrator component, communicating with the same number of interactions as P1.
- P3: we consider the same set of interactions as P1 and P2 handled with four orchestrator components (two for each level), assuring communication between a set of 10 S/R components.

The experiments are run to calculate execution time for all system scenarios, first without introducing security mechanisms on interactions and then after adding them. From Figure 14, we can see that the architecture of the system model have no significant effect on the execution time of the model where executing the same binding set between components do not introduce a significant overhead. The use of cryptographic mechanisms induces an average overhead of 20%, however this performance can be improved if we choose to use, for instance,

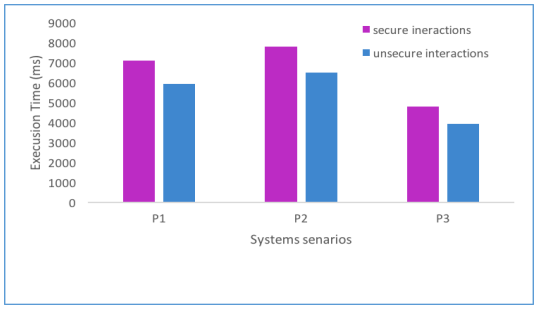

Figure 14: Execution time with different scenarios for the When_App application.

symmetric encryption instead of the asymmetric one currently implemented. In scenario P3, rising the number of orchestrator components allows more concurrency to execute components and interactions which reduces the execution time of the system.

There exists diverse authentication and encryption mechanisms used to sign messages by including the corresponding tokens into the security header of the message and encode them. The choice between simple tokens (User-name/Clear Password, User-name/Password Digest), Binary Tokens (X.509 certificates, Kerberos) or XML Tokens (SAML assertions, XrML (eXtensible Rights Markup Language), XCBF (XML Common Biometric Format)) for signature and encryption algorithms is depending on the application context and the required quality of service.

We also emphasize that several works on model-based security aim at simplifying security configuration and coding [24, 25]. In [25], authors, propose modeling security policy in UML and target automating security code generation for business applications like JEE and .net applications. Other works [24] use model-based approach to simplify secure code deployment on heterogeneous platforms. Compared to these, our work is not restricted to point-to-point access control and deals with information flow security. If we consider the *Whens_App* application, it is not clear how these tools can manage multiparty interactions while preserving information and participants privacy. Recent works on information flow security in web services, rely on Petri-nets for modeling composed services [26]. First, Petri-net graphs are generated from *BPEL* orchestration processes and are, next, modified by the developer to represent shared resources and to annotate interactions. Developers modification is necessary here since Petri-nets capture event-based interactions only. Our model allows representing both data and events.

6. Related Work

BPEL decentralisation is to decompose a BPEL process into several subprocesses, each deployed and executed by a different orchestrator. To do this, in

[27], authors propose to use techniques for analyzing and re-scheduling nodes to minimize the communication load in case several instances are executed at the same time using workflow dependency graphs. The same work is extended to discuss different aspects of decentralization such as synchronization problems in the case of decentralized execution, or the implementation of restrictions on the flow of data between translated fragments. More recently, Lifeng et al. [28] proposed an extension of this approach to decentralization of BPEL processes. This extension concerns the optimization of the partitioning process using a genetic algorithm. First they create an initial partitioning topology, then they apply transformations using genetic operators (selection, crossing and mutation) on this solution. Then, they evaluate the new solution and reiterate until reaching a threshold on the number of iterations. The partitioning chosen is the one that has a better quality (using an evaluation function: adaptation function). However, these two approaches do not provide a generic partitioning methodology independent of the composition language, nor consider the distribution constraints (collocate and separate). Hence, the designer has no control over the decentralization process. In [29, 30], Yildiz et al. Consider the decentralization process in an abstract way and extend the deadline elimination algorithm used by the *BPEL* process execution engines. Their contribution seeks to preserve the constraints of the control flow of the centralized specification, and to prevent a deadlock in interactions between services. Most of the techniques developed address particular aspects of decentralization, rather than providing a generic and flexible methodology. The major disadvantage of these approaches is their dependencies of the specification language. In our work, we propose a secure-by-construction approach that handles a decentralized orchestration of complex systems with multi-party interactions. This approach is automated, language independent and practical which takes also handles information flow security.

Model-based security aims at simplifying security configuration and coding. The work in [25] considers modelling security policies in UML and targets automating security code generation for business applications using JEE and .net. The work of [24] uses a model-based approach to simplify secure code deployment on heterogeneous platforms. Compared to these, our work is not restricted to point-to-point access control and deals with information flow security. The work on designing web services from [31] relies on Petri-nets for modelling composed services and annotations for the flow of interactions. Our component model is more general and deals with both data and event- non-interference.

Information flow control for programming languages dates back to Denning who originally proposed a language for static information flow checking [32]. Since then, information-flow control based on type systems and associated compilation tools has widely developed [33, 34, 35]. Recently, it extends to provably-secure languages including cryptographic functions[36, 37, 38, 39, 40]. With few exceptions, all these approaches are restricted to sequential imperative languages and ignore distribution/communication aspects. Among the exceptions, JifSplit [41] takes as input a security-annotated program, and splits it

into threads by assuming that the communication through the network is secure. Furthermore, in [42] the communication's security is enforced by adding cryptographic mechanisms. The drawback of these is that the security aspect guides the system distribution. In practice, a separation of concerns is required and the system architecture must be independent of security constraints. Our approach is different since our starting point is a component-based model and the security constraints are expressed with annotations at the architecture level.

Operating systems like Flume [43], HiStar [44] and Asbestos [45] ensure information flow control between processes by associating security labels to processes and messages. DStar [46] extends HiStar to distributed applications. These approaches may appear attractive since transparent to the developer. Nevertheless, the granularity of processes may be too coarse to establish end-to-end security for distributed applications with complex interactions.

Component-based design is appealing for verification of security since the system structure and communications are explicitly represented. However, existing work focus merely on point-yo-point access control. The work of [47] considers dependencies between service components but not advanced properties like implicit information flow. In [48], authors provide APIs to configure the security of component connectors. The work in [49] deals with non-interference on component-based models using annotation propagation inside component code. In our work, we achieve complete separation between the abstract high-level component model on which non-interference is verified, and the low-level platform-dependent model where security is enforced by construction.

7. Conclusion and Future Work

We introduced a tool-supported approach to automatically orchestrate and secure information flow in composed WS. By abstracting the system behavior to a component-based model with multi-party interactions, we verify security policy preservation, that is, non-interference property at both event and data levels. Then, we generate a distributed model where multi-party interactions are replaced with protocols based on the use message passing. The distributed model is proved "secure-by-construction". As a target for the S/R distributed model, we generate a set of orchestrated *BPEL* processes that relying on web services security standards, ensure the required protection of the information flow. On longer term, we plan to extend both the security model and the associated transformations for relaxed versions of non-interference i.e, allowing runtime re-labelling, declassification, intransitive.

References

[1] Walsh, A.: UDDI, SOAP, and WSDL: The Web Services Specification Reference Book. Prentice Hall (2002)

[2] Juric, M.B.: Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition. Packt Publishing 2006 (2006)

[3] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf

[4] Guillou, X.L., Cordier, M.O., Robin, S., Roze, L.: Monitoring ws-cdl-based choreographies of web services. In: Proceedings of the 20th International Workshop on Principles of Diagnosis. (June 2009) 43–50

[5] Damiani, E., di Vimercati, S.D.C., Paraboschi, S., Samarati, P.: Securing SOAP e-services. International Journal of Information Security **1**(2) (2002) 100–115

[6] Della-Libera, G., Gudgin, M., Hallam-Baker, P., Hondo, M., Granqvist, H., Kaler, C., Maruyama, H., McIntosh, M., Nadalin, A., Nagaratnam, N., Philpott, R., Prafullchandra, H., Shewchuk, J., Walter, D., Zolfonoon, R.: Web services security policy language (WS-SECURITYPOLICY). Technical report (2005)

[7] Sabelfeld, A., Sands, D.: A per model of secure information flow in sequential programs. Higher Order Symbolic Computation **14**(1) (2001) 59–91

[8] Smith, G., Volpano, D.: Secure information flow in a multi-threaded imperative language. In: Symposium on Principles of Programming Languages (POPL'98), ACM (1998) 355–364

[9] Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE Journal on Selected Areas in Communications **21**(1) (2003)

[10] McCullough, D.: Noninterference and the composability of security properties. In: Security and Privacy (SP'88), IEEE Computer Society (1988) 177–186

[11] McLean, J.: A general theory of composition for trace sets closed under selective interleaving functions. In: Security and Privacy (SP'94), IEEE Computer Society (1994) 79

[12] Zakinthinos, A., Lee, E.S.: A general theory of security properties. In: Security and Privacy (SP'97), IEEE Computer Society (1997) 94–102

[13] Mantel, H.: Possibilistic Definitions of Security - An Assembly Kit. In: 13th IEEE Workshop on Computer Security Foundations (CSFW'00), IEEE Computer Society (2000) 185

[14] Said, N.B., Abdellatif, T., Bensalem, S., Bozga, M.: A model-based approach to secure multiparty distributed systems. In: Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part I. (2016) 893–908

[15] Ben Said, N., Abdellatif, T., Bensalem, S., Bozga, M.: A robust framework for securing composed web services. In: FACS'15, Revised Selected Papers. Volume 9539 of LNCS., Springer (2016) 105–122

[16] Ben Said, N., Abdellatif, T., Bensalem, S., Bozga, M.: Model-driven information flow security for component-based systems. In: ETAPS/FPS'14 Proceedings. Volume 8415 of LNCS., Springer (2014) 1–20

[17] Accorsi, R., Lehmann, A.: Automatic information flow analysis of business process models. In: BPM'12 Proceedings. Volume 7481 of LNCS., Springer (2012) 172–187

[18] Focardi, R., Rossi, S., Sabelfeld, A.: Bridging language-based and process calculi security. In: FOSSACS'05 Proceedings. Volume 3441 of LNCS., Springer (2005) 299–315

[19] Frau, S., Gorrieri, R., Ferigato, C.: Petri net security checker: Structural non-interference at work. In: FAST'08 Proceedings. Volume 5491 of LNCS., Springer (2009) 210–225

[20] Bonakdarpour, B., Bozga, M., Jaber, M., Quilbeuf, J., Sifakis, J.: Automated conflict-free distributed implementation of component-based models. In: SIES'10 Proceedings, IEEE (2010) 108–117

[21] Zdancewic, S., Zheng, L., Nystrom, N., Myers, A.C.: Secure program partitioning. ACM Trans. Comput. Syst. (2002)

[22] van der Aalst, W.M.P., Lassen, K.B.: Translating workflow nets to bpel. Technical report, techreport (2005)

[23] Myers, A.C., Liskov, B.: Protecting privacy using the decentralized label model. ACM Transactions on Software Engineering and Methodology **9** (2000)

[24] Chollet, S., Lalanda, P.: Security specification at process level. In: SCC'08 Proceedings, IEEE Computer Society (2008) 165–172

[25] Basin, D.A., Doser, J., Lodderstedt, T.: Model driven security: From UML models to access control infrastructures. ACM Trans. Softw. Eng. Methodol. **15**(1) (2006) 39–91

[26] Accorsi, R., Wonnemann, C.: Static information flow analysis of workflow models. In: Conference on Business Process and Service Computing, volume 147 of Lecture Notes in Informatics. (2010) 194–205

[27] Goettelmann, E.: Risk-aware Business Process Modelling and Trusted Deployment in the Cloud. Theses, Université de Lorraine (October 2015)

[28] Lifeng, A., Maolin, T., Colin, F.: Partitioning composite web services for decentralized execution using a genetic algorithm. Future Gener. Comput. Syst. (2011)

[29] Yildiz, U., Godart, C.: Centralized versus decentralized conversation-based orchestrations. In: 9th IEEE International Conference on E-Commerce Technology (CEC 2007) / 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (EEE 2007), 23-26 July 2007, National Center of Sciences, Tokyo, Japan. (2007) 289–296

[30] Yildiz, U., Godart, C.: Information flow control with decentralized service compositions. 2007 IEEE International Conference on Web Services **00** (2007) 9–17

[31] Accorsi, R., Wonnemann, C.: Static information flow analysis of workflow models. In: ISSS and BPSC'10 Proceedings. Volume 177 of LNI. (2010) 194–205

[32] Denning, D.E., Denning, P.J.: Certification of programs for secure information flow. Commun. ACM (1977) 504–513

[33] Goguen, J., A. Meseguer, J.: Security policies and security models. In: 1982 IEEE symposium on Security and Privacy, IEEE Computer Society (1982) 11–20

[34] Heintze, N., Riecke, J.G.: The slam calculus: Programming with secrecy and integrity. In: POPL'98 Proceedings, ACM (1998) 365–377

[35] Volpano, D.M., Irvine, C.E., Smith, G.: A sound type system for secure flow analysis. Journal of Computer Security **4**(2/3) (1996) 167–188

[36] Laud, P.: Semantics and program analysis of computationally secure information flow. In: ESOP'01 Proceedings. Volume 2028 of LNCS., Springer (2001) 77–91

[37] Adão, P., Fournet, C.: Cryptographically sound implementations for communicating processes. In: ICALP'06 Proceedings. Volume 4052 of LNCS., Springer (2006) 83–94

[38] Courant, J., Ene, C., Lakhnech, Y.: Computationally sound typing for non-interference: The case of deterministic encryption. In: FSTTCS'07 Proceedings. Volume 4855 of LNCS., Springer (2007) 364–375

[39] Laud, P.: On the computational soundness of cryptographically masked flows. In: POPL'08 Proceedings, ACM (2008) 337–348

[40] Fournet, C., Rezk, T.: Cryptographically sound implementations for typed information-flow security. In: POPL'08 Proceedings, ACM (2008) 323–335

[41] Zdancewic, S., Zheng, L., Nystrom, N., Myers, A.C.: Secure program partitioning. ACM Trans. Comput. Syst. (2002) 283–328

[42] Fournet, C., Le Guernic, G., Rezk, T.: A security-preserving compiler for distributed programs: From information-flow policies to cryptographic mechanisms. In: CCS'09 Proceedings, ACM (2009) 432–441

[43] Krohn, M.N., Yip, A., Brodsky, M.Z., Cliffer, N., Kaashoek, M.F., Kohler, E., Morris, R.: Information flow control for standard OS abstractions. In: SOSP'07 Proceedings, ACM (2007) 321–334

[44] Zeldovich, N., Boyd-Wickizer, S., Kohler, E., Mazières, D.: Making information flow explicit in HiStar. In: OSDI'06 Proceedings, Usenix Assoc. (2006) 263–278

[45] Vandebogart, S., Efstathopoulos, P., Kohler, E., Krohn, M.N., Frey, C., Ziegler, D., Kaashoek, M.F., Morris, R., Mazières, D.: Labels and event processes in the Asbestos operating system. ACM Trans. Comput. Syst. **25**(4) (2007)

[46] Zeldovich, N., Boyd-Wickizer, S., Mazières, D.: Securing distributed systems with information flow control. In: NSDI'08 Proceedings, Usenix Assoc. (2008) 293–308

[47] Parrend, P., Frénot, S.: Security benchmarks of OSGi platforms: toward hardened OSGi. Softw., Pract. Exper. **39**(5) (2009) 471–499

[48] Kuz, I., Liu, Y., Gorton, I., Heiser, G.: Camkes: A component model for secure microkernel-based embedded systems. Journal of Systems and Software **80**(5) (2007) 687–699

[49] Abdellatif, T., Sfaxi, L., Robbana, R., Lakhnech, Y.: Automating information flow control in component-based distributed systems. In: CBSE'11 Proceedings, ACM (2011) 73–82