A Model-Based Approach to Secure Multiparty Distributed Systems

Najah Said, Takoua Abdellatif, Saddek Bensalem, Marius Bozga

A Model-Based Approach to Secure Multiparty Distributed Systems *

Najah Ben Said[1], Takoua Abdellatif[2], Saddek Bensalem[1], and Marius Bozga[1]

[1] Univ. Grenoble Alpes, VERIMAG, F-38000 Grenoble, France
CNRS, VERIMAG, F-38000 Grenoble, France
[2] Tunisia Polytechnic School, University of Carthage, Tunis, Tunisia

Abstract. Within distributed systems with completely distributed interactions between parties with mutual distrust, it is hard to control the (illicit) flowing of private information to unintended parties. Unlike existing methods dealing with verification of low-level cryptographic protocols, we propose a novel model-based approach based on model transformations to build a secure-by-construction multiparty distributed system. First, starting from a component-based model of the system, the designer annotates different parts of it in order to define the security policy. Then, the security is checked and when valid, a secure distributed model, consistent with the desired security policy, is automatically generated. To illustrate the approach, we present a framework that implements our method and use it to secure an online social network application.

1 Introduction

Model-based development aims at both reducing development costs and increasing the integrity of system implementations by using explicit models employed in clearly defined transformation steps leading to correct-by-construction implementation artifacts. 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.

Nonetheless, ensuring end-to-end security requirements in distributed systems remains a difficult and error-prone task. In many situations, security reduces 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 and deciding how to set access control permissions in complex systems is a difficult problem in itself. Equally, using cryptographic primitives that provides strong

* 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).

confidentiality and integrity guarantees, is also less helpful to ensure that the system obeys an overall security policy.

Information flow control is a much robust alternative which tracks information propagation in the entire system and prevent secret or confidential information from being publicly released. Information flow control relies on annotating system data and/or actions with specific levels of security and use specific methods for checking non-interference, that is, absence of leakage, between different security levels. Nonetheless, providing annotations and establishing their correctness is equally difficult especially for distributed implementations, where only code is available and no higher-level abstractions.

In this paper we introduce a model-based development approach for dealing with information flow control in distributed systems. Our contribution can be summarized as follows. First, we provide a high-level component-based model and associated security annotations to allow system designers to configure the system security in an intuitive way. They do not need to be experts in security or cryptographic protocols. Second, we provide a security checker that applies information flow techniques and verifies formally the correctness of the provided configuration. Third, a distributed model is automatically generated and respects the designer configured security policy. The mapping between the high-level policy and the distributed model is formally proven.

In the paper, we use the *secureBIP* framework [1] as underlying component-based modeling framework. *secureBIP* is an extension of the *BIP* framework [2] with information flow security. Given security annotations for data and interactions, *secureBIP* captures two types of non-interference, respectively *event* and *data non-interference*. For events (that is, occurrences of interactions), non-interference states that the observation of public events does not reveal any information about the occurrence of secret events. For data, it states that there is no leakage of secret data into public ones.

We provide model transformations allowing to transform high-level *secure-BIP* models into a distributed models while preserving event and data non-interference. In this way, information flow security needs to be verified once, for the high-level model, and then it holds *by construction* on the distributed model and later on the final implementation. The proposed transformation extends previous work [3,4] on distributed implementation of *BIP* components models, which essentially addressed functional and performance aspects, while being totally agnostic about security-related issues.

The paper is structured as follows. Section 2 presents a running example to be used along the paper. Section 3 recalls the main concepts of the *secureBIP* framework, associated non-interference definitions and security conditions. Next, section 4 contains the new automated distribution approach to derive secure distributed models. Finally, section 5 discusses related work and section 6 concludes and presents some perspectives for future research. Proofs of technical results are given in a technical report[3].

[3] no. TR-2014-6 on http://www-verimag.imag.fr/Rapports-Techniques,28.html

2 Running Example

Throughout the paper, we consider a simplified social network application, called Whens-App, and illustrated in Figure 1. The application is intended for organizing virtual events where participants can meet and exchange data.

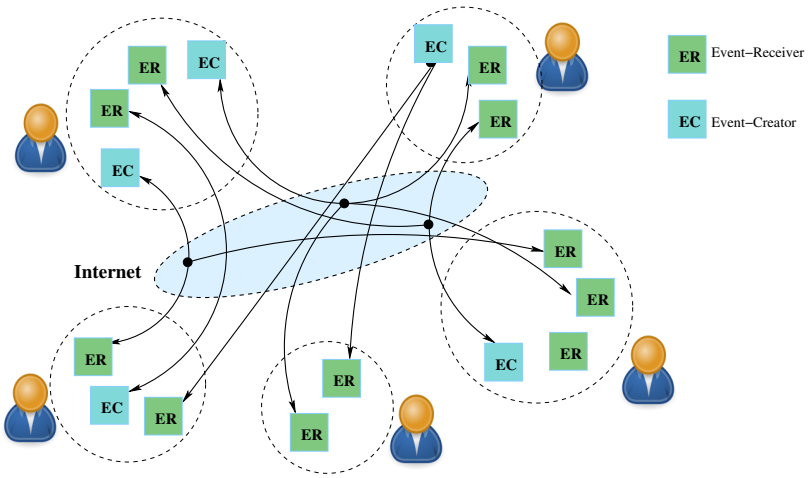

Fig. 1. Overview of the Whens-App application

As social network application, Whens-App entails several of security requirements. In this paper we focus on requirements related to information flow security: assuming that components are trustful and the network is unsecure, *(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 the annotated model can be automatically and systematically transformed towards a distributed implementation while preserving the security properties.

3 Secure Component 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). *An atomic component B is a tuple (L, X, P, T) where L is a set of locations, X is a set of variables, P is a set*

of ports and $T \subseteq L \times P \times L$ *is a set of port labelled transitions. For every port* $p \in P$*, we denote by* X_p *the subset of variables exported and available for interaction through* p*. For every transition* $\tau \in T$*, we denote by* g_τ *its guard, that is, a Boolean expression defined on* X *and by* f_τ *its update function, that is, a parallel assignment* $\{x := e_\tau^x\}_{x \in X}$ *to variables of* X*.*

Let $\mathcal{D}$ be an 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 $\text{SEM}(B) = (Q_B, \Sigma_B, \underset{B}{\longrightarrow})$ where the set of states $Q_B = L \times \mathbf{X}$, the set of labels $\Sigma_B = P \times \mathbf{X}$ and transitions $\underset{B}{\longrightarrow}$ are defined by the rule:

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

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.

Figure 2 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]

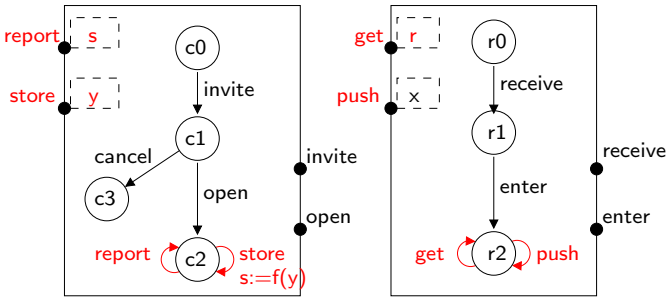

Fig. 2. Example of atomic components

Composite components are obtained by composing atomic components $B_i = (L_i, X_i, P_i, T_i)_{i=1,n}$ through multiparty 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 multiparty *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., $\bigcup_{p \in P_a} X_p$).

Definition 2 (composite component). *A composite component $C = \gamma(B_1, \ldots, B_n)$ consists of the composition of $B_1, \ldots, B_n$ by a set of interactions γ.*

Given $\text{SEM}(B_i) = (Q_i, \Sigma_i, \xrightarrow[B_i]{})_{i=1,n}$, the semantics of C is defined as 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 transitions $\xrightarrow[C]{}$ are defined by the rule:

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

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 C 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 $\mathbf{x}_{p_i}$ of exported variables on ports in a. Execution of a triggers first the data transfer function F_a which modifies exported variables X_{p_i}. 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.

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 $\text{TRACES}(C, q_0)$.

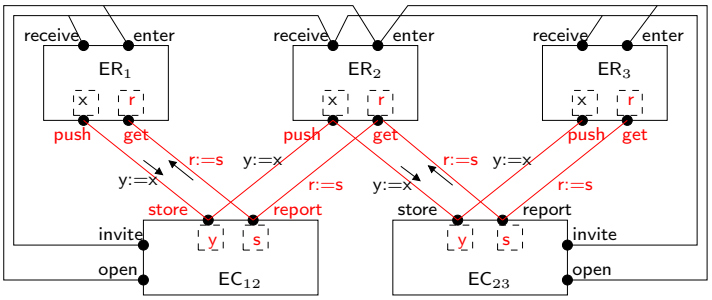

Fig. 3. Example of composite component

Figure 3 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 $\langle \{L, H\}, \{(L, L), (L, H), (H, H)\} \rangle$.

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

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

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

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 occurences 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 2 and 3 is annotated with two levels of security *Low* (in black) and *High* (in red). With this assignement, 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 [1], 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-interfrence 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 [5] for excluding causal and conflicting places for Petri net transitions having different security levels. Similar conditions have been considered in [6,7] and lead to more specific definitions of non-interferences and bisimulations 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 [1], 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 2 and 3. 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 Automatic Decentralization Method

In this section, we describe the decentralization method for our component-based model and provide formal proofs for information-flow security preservation. The decentralization method extends the method for decentralization of BIP models [3,4]. The existing method transforms BIP models with multiparty interactions (and priorities) into functionally equivalent BIP models using only send/receive (S/R) interactions. S/R interactions are binary, point-to-point, directed interactions from one sender component (port), to one receiver component (port) implementing asynchronous message passing.

From a functional viewpoint, the main challenge when transforming a BIP model into a decentralized S/R BIP model is to enable parallelism for execution of atomic components and concurrently enabled interactions. That is, in a distributed setting, every atomic component executes independently and communication is restricted to asynchronous message passing. The existing method for decentralizing BIP relies on structuring the distributed components according to a hierarchical architecture with two[4] layers:

- the *atomic components* layer includes transformed atomic components. Whenever an atomic component needs to interact, it publish an offer, that is the list of its enabled ports, then wait for a notification indicating which interaction has been chosen, and then resume its execution.
- the *interaction protocols (IP)* layer deals with distributed execution of interactions by implementing specific protocols. Every IP component handles a subset of interactions, that is, check them for enabledness and schedule them for execution accordingly. The interface between this layer and the component layer provides ports for receiving offers and notifying the ports selected for execution.

[4] In general, a third *conflict resolution* layer is used, however, it has a confined impact on information flow and is ommited here for the sake of simplicity of the presentation

The existing methods in [3,4] have been designed without taking into account security concerns. In the following, we will show that they can be extended such that to preserve information flow security. Roughly speaking, this is achieved by using a slightly different transformation for atomic components as well as by imposing few additional restrictions on the structure of the interaction protocol layer. 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 Layer

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 to some IP component and the notification transition corresponds to receiving the answer from an IP component, once an interaction has been completed. Update functions can be then executed concurrently and independently by components upon notification reception.

In constrast to the transformation proposed in [3], several changes are needed to protect information flow. Distinct offer ports o_s and interaction counters n_s are introduced for every security level. 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 trasformed component.

Definition 7 (security assignement 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 assignement for the Event Receiver are depicted in Figure 4. 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 assignement obeys all the (local) security conditions related to B^{SR}.

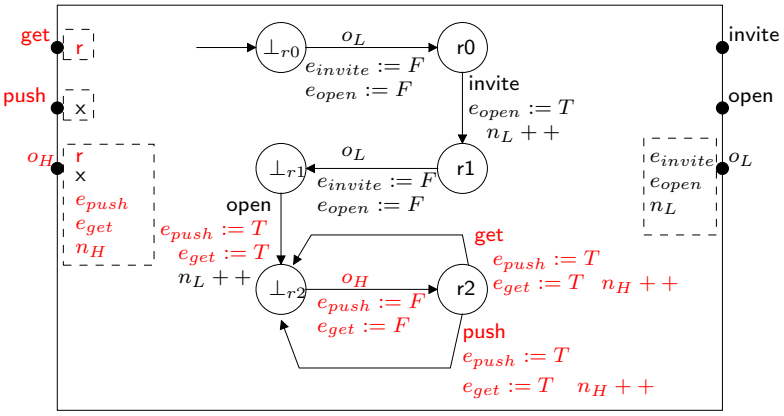

Fig. 4. Transformation of atomic components illustrated on the Event Receiver

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

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

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

4.2 Interaction Protocol Layer

This layer consists of a set of components, each in charge of execution of a subset of interactions from the original component model. Every such IP 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 IP components handling a conflict-free partitioning of interactions, as in [3]. 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 IP components to run fully independently of each other, that is, local decisions taken on every IP 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 partitioning, that is, the subset of interactions handled within every IP component must have the same security level. Intuitively, this restriction allows us to enforce by construction the security conditions for all IP components 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 IP component IP_s handling the subset of interactions $\gamma_s = \{a \in \gamma \mid \sigma(a) = s\}$ with security level s.

Definition 8 (IP_s component for γ_s). *Interaction protocol component IP_s handling interactions γ_s is defined according to [3], Definition 7.*

The extended security assignement σ^{SR} for IP_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 IP_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 IP_s.

Definition 9 (security assignment σ^{SR} for IP_s). *The security assignment σ^{SR} is built from the original security assignment σ. For variables X^{IP} and ports P^{IP} of the IP_s component that handles γ_s, we define*

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

The above definition enforces the security conditions for IP_s.

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

Proof. Trivial check for conditions (i, iv). The condition (ii) on sequential consistency is also valid, even if some (replicated) variables within IP_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.

4.3 System Composition

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

- for every component B_i^{SR} participating in interactions having security level s, include in γ^{SR} the *offer* interaction $(B_i^{SR}.o_s, IP_s.o_i)$ associated with the transfer of data from the component port o_s to the IP component port o_i.
- for every port p in component B_i^{SR} with security level s, include in γ^{SR} the *notification* interaction $(IP_s.p, B_i.p)$ associated with the transfer of the subset of X_p variables having security level at least s from the IP 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 IP_s has security level s. The construction is illustrated for the running example in Figure 5. 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 IP_H and not sent back on the notification of the push interaction.

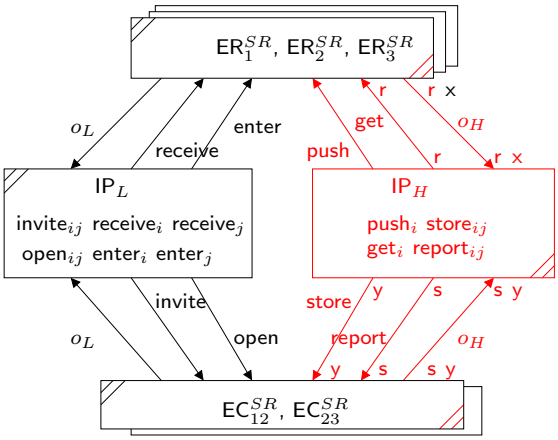

Fig. 5. Decentralized model for the WhensApp example

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* $C^{SR} = \gamma^{SR}(B_1^{SR}, ..., B_n^{SR}, (IP_s)_{s\in S})$ *satisfies security conditions for the security assignment* σ^{SR}.

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

5 Related Work

Model-based security aims at simplifying security configuration and coding. The work in [8] considers modeling security policies in UML and targets automating security code generation for business applications using JEE and .net. The work of [9] 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 [10] relies on Petri-nets for modeling 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 [11]. Since then, information-flow control based on type systems and associated compilation tools has widely developed [12,13,14]. Recently, it extends to provably-secure languages including cryptographic functions[15,16,17,18,19]. With few exceptions, all these approaches are restricted to sequential imperative languages and ignore distribution/communication aspects. Among the exceptions, JifSplit [20] takes as input a security-annotated program, and splits it into threads by assuming that the communication through the network is secure. Furthermore, in [21] 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 [22], HiStar [23] and Asbestos [24] ensure information flow control between processes by associating security labels to processes and messages. DStar [25] 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 [26] considers dependencies between service components but not advanced properties like implicit information flow. In [27], authors provide APIs to configure the security of component connectors. The work in [28] 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.

6 Conclusion and Future Work

We introduced a tool-supported approach to automatically secure information flow in distributed systems. Starting from an abstract component-based model with multiparty interactions, we verify security policy preservation, that is, non-interference property at both event and data levels. Then, we generate a distributed model where multiparty interactions are replaced with protocols based on asynchronous message passing. The distributed model is proved "secure-by-construction". This work is being extended towards code generation and deployment on distributed platforms. More specifically, we envisage to use web services as a target for the S/R distributed model and to rely on web services security standards to ensure the required protection of the information flow, following idea from [29]. On longer term, we plan to 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. 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
2. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time systems in BIP. In: SEFM'06 Proceedings, IEEE Computer Society Press (2006) 3–12
3. 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
4. Bonakdarpour, B., Bozga, M., Jaber, M., Quilbeuf, J., Sifakis, J.: A framework for automated distributed implementation of component-based models. Distributed Computing **25**(5) (2012) 383–409
5. Accorsi, R., Lehmann, A.: Automatic information flow analysis of business process models. In: BPM'12 Proceedings. Volume 7481 of LNCS., Springer (2012) 172–187
6. 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
7. 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
8. 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
9. Chollet, S., Lalanda, P.: Security specification at process level. In: SCC'08 Proceedings, IEEE Computer Society (2008) 165–172

10. Accorsi, R., Wonnemann, C.: Static information flow analysis of workflow models. In: ISSS and BPSC'10 Proceedings. Volume 177 of LNI. (2010) 194–205
11. Denning, D.E., Denning, P.J.: Certification of programs for secure information flow. Commun. ACM (1977) 504–513
12. Goguen, J., A. Meseguer, J.: Security policies and security models. In: 1982 IEEE symposium on Security and Privacy, IEEE Computer Society (1982) 11–20
13. Heintze, N., Riecke, J.G.: The slam calculus: Programming with secrecy and integrity. In: POPL'98 Proceedings, ACM (1998) 365–377
14. 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
15. Laud, P.: Semantics and program analysis of computationally secure information flow. In: ESOP'01 Proceedings. Volume 2028 of LNCS., Springer (2001) 77–91
16. Adão, P., Fournet, C.: Cryptographically sound implementations for communicating processes. In: ICALP'06 Proceedings. Volume 4052 of LNCS., Springer (2006) 83–94
17. Courant, J., Ene, C., Lakhnech, Y.: Computationally sound typing for noninterference: The case of deterministic encryption. In: FSTTCS'07 Proceedings. Volume 4855 of LNCS., Springer (2007) 364–375
18. Laud, P.: On the computational soundness of cryptographically masked flows. In: POPL'08 Proceedings, ACM (2008) 337–348
19. Fournet, C., Rezk, T.: Cryptographically sound implementations for typed information-flow security. In: POPL'08 Proceedings, ACM (2008) 323–335
20. Zdancewic, S., Zheng, L., Nystrom, N., Myers, A.C.: Secure program partitioning. ACM Trans. Comput. Syst. (2002) 283–328
21. 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
22. 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
23. Zeldovich, N., Boyd-Wickizer, S., Kohler, E., Mazières, D.: Making information flow explicit in HiStar. In: OSDI'06 Proceedings, Usenix Assoc. (2006) 263–278
24. 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)
25. Zeldovich, N., Boyd-Wickizer, S., Mazières, D.: Securing distributed systems with information flow control. In: NSDI'08 Proceedings, Usenix Assoc. (2008) 293–308
26. Parrend, P., Frénot, S.: Security benchmarks of OSGi platforms: toward hardened OSGi. Softw., Pract. Exper. **39**(5) (2009) 471–499
27. 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
28. 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
29. 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