A Model-based Approach to Secure Multi-party Distributed Systems

Najah Ben Said, Saddek Bensalem, Marius Bozga
Univ. Grenoble Alpes, VERIMAG, F-38000 Grenoble, France
CNRS, VERIMAG, F-38000 Grenoble, France
FirstName.LastName@imag.fr

Takoua Abdellatif
University of Carthage, Tunis
Tunisia Polytechnic School, Tunisia
takoua_abdellatif@yahoo.fr

ABSTRACT

Securing multi-party distributed systems is still a challenge. In such 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 some existing solutions dealing with verification of low-level cryptographic protocol in multi-party interactions, we propose a novel approach based on model transformations to build secure-by-construction multi-party distributed systems. The user has to describe his system in a component-based model and annotate it to define the system security policy. Then, the system is checked and when valid, a secure code, consistent with the desired security policy, is automatically generated. To validate the approach, we present a framework that implements our method and we use it to secure an online social network application.

Keywords

component-based systems, distributed systems, model transformation, information flow security

1. INTRODUCTION

Multi-party distributed systems involve different parties and interactions with generally mutual distrust and unsecure underlying communication channels. In such systems, it is still challenging to protect people privacy and make sure that classified information are only disclosed to intended parties. The typical example is the online social network (OSN) where members within groups exchange information and events and do not necessarily control how their data are disseminated within OSN users and storage. Furthermore, it is difficult to control when some public information implicitly reveal some secret one (for instance, a travel ticket price can disclose the travel destination). Indeed, many studies on OSN such as Facebook show that the security configuration settings fall short to ensure intended policies. Another example of multi-party application is the Health-Service-App, used to calculate patients profiles and make statistics in collaboration with a set of distributed hospital applications. It is obvious that some parties are not allowed to access all health private details of patients. Considering an other example of data-mining, privacy preservation is rather treated by a cryptograph-based research known as multi-party secure computation [12]. In this field, researchers adopt formal frameworks for verifying low-level adopted cryptographic protocols.

In this work, we are interested in verifying multi-party security systems at an abstract level model before even defining protocols used in the system. The advantage of such approach is that the system designer can describe separately his system component behavior and interactions at a high-level and then configures security in an intuitive way. Then, the designer automatically checks the intended security policy without worrying about the burden of the application protocol details. Hence, we guarantee that security specifications, that can be modified within the system life cycle, are treated independently and in parallel with functional specifications. For security checking, we adopt the non-interference property [11] to enforce privacy and make sure that information are not leaking in an explicit or implicit way.

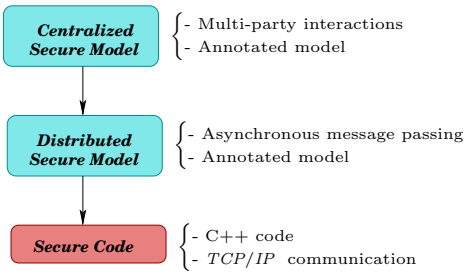

Figure 1: Model Transformation

In this paper, we present a practical automated method to build secure-by-construction distributed systems using a model-based approach. Starting from a multi-party centralized model, the system is described as a set of components and interactions. Annotations at the level of variables and interfaces of components and interactions allow configuring the system security policies. The centralized model is transformed to a decentralized send/receive (S/R) model that is, scheduling and communication protocol components are inserted to implement interactions and set-up distributed communication protocols. Security annotations are propagated to the new distributed component-based architecture following a set of rules to preserve the non-interference property. In the last step, the S/R model is used to produce the distributed code. The model transformations are illustrated in Figure 1. To the best of our knowledge, this is the first approach to securely decentralize high-level component-based systems with multiparty interactions. We proved that, whenever the input model is secure, that is, satisfies conditions for event and data non-interference of [5] , the decen-

tralized model is also secure. The first transformation has been designed such that to preserve, by construction, the non-interference property. For the code generation, the implementation is directly derived from the S/R model using secure communication primitives. Our contribution can be summarized in the following points:

- We define an automated method based on model transformations to build a secure-by-construction multi- party distributed system; the user has only to design his system in a component-based model with multiparty interactions.
- We provide formal definitions of non-interference for component-based models and we correctness proofs of different transformation steps by showing the preservation of non-interference property. We define two kinds of non-interference: event and data non-interference for a more rigorous and fine-grained verification in distributed systems.
- We present a framework that implements our method and use it to secure an OSN application called *Whens-App* for event organization. This application is general enough to cover many multi-party applications with different security configurations.

The paper is structured as follows. Section 2 introduces the *Whens-App* application and motivates the need for our work and approach. Section 3 presents the main concepts of the component-based framework adopted in this work as well as non-interference definitions and sufficient conditions. In section 4, we describe the automated distribution approach to derive secure executable code. Section 5 presents the implementation Tool-set and evaluation section 6 discusses the related work. The section 7 concludes and presents some perspectives for future work. All proofs of technical results are given in a technical report[1].

2. CASE STUDY

Throughout the paper, we consider *Whens-App*, an OSN application for organizing events, such as business meeting where participants can exchange data when these meetings take place. Figure 2 shows an overview of a fragment of the system which consists of a finite number of *Event-Creators*, each communicating with a set of *Event-Receivers*. Communication channels are represented by lines in the figure.

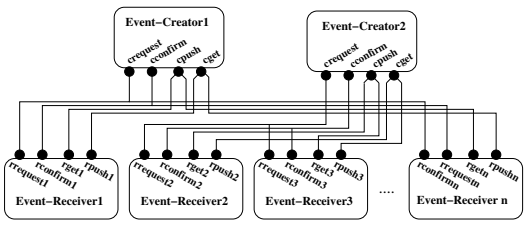

Figure 2: High-level description of the Whens-App system.

As social network application, *Whens-App*, entails a large variety of security requirements, in this paper however, we focus on some relevant 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 does not reveal any information about event participants and *(2)* confidentiality of classified data as well as events is always preserved and kept secret inter- as well as intra-components. Both requirements are ensured by using security annotations model for tracking events and data in the system and checking that the formal model satisfies the security constraints given in Section 3.3. We additionally enforce privacy of participants at implementation by hiding the identity of components participating in a secret interaction.

[1]http://www-verimag.imag.fr/Technical-Reports,264.html?lang=en&-number=TR-2014-6

3. SECURE COMPONENT-BASED MODEL

The centralized secure model [5] presents a component oriented model well adapted in describing complex systems like heterogeneous and distributed ones. It has been carefully made to achieve high expressivity for component composition, while maintaining separation of concerns between components behavior and their coordination. Thanks to its modularity, it offers a flexible way to develop and manage complex systems. Particularly, for information flow security, the explicit system architecture it provides allows tracking easily intra and inter-components information flow. Furthermore, the centralized model allows the development of scalable validation and verification methods and tools, exploiting compositionality principles. We recall hereafter the main concepts behind component model with a particular focus on security annotations and the different notions of non-interference and their verification.

3.1 Component-Based Model

The system functional model is expressed as a set of atomic components, that is, finite state automata or 1-safe Petri nets, extended with data. Communications inter-components are achieved using interactions that express synchronization constraints and do the transfer of data between the interacting components. In the following, we recall the key concepts of the used component-based model which are further relevant for dealing with information flow security. In particular, we give a formal definition of atomic components and their composition through multiparty interactions.

DEFINITION 1 (ATOMIC COMPONENT). *An atomic component B is a tuple (L, X, P, T) where L is a set of states, 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 $\{x := e^x_\tau\}_{x \in X}$ to variables of X.*

Let $\mathcal{D}$ be the data domain of variables. Given a set of variables Y, we call valuation on Y any function $\mathbf{y} : Y \to \mathcal{D}$ mapping variables to data. We denote by $\mathbf{Y}$ the set of all valuations defined on Y. The semantics of an atomic component $B = (L, X, P, T)$ is defined as the labelled transition system $\text{LTS}(B) = (Q_B, \Sigma_B, \underset{B}{\longrightarrow})$ where the set of states $Q_B = L \times \mathbf{X}$, the set of labels is $\Sigma_B = P \times \mathbf{X}$ and the set of

labelled transitions $\xrightarrow[B]{}$ is 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 valuation $\mathbf{x}$, (3) $\mathbf{x}''_p$ is a valuation of exported variables X_p and (4) $\mathbf{x}' = f_\tau(\mathbf{x}[X_p \leftarrow \mathbf{x}''_p])$ meaning that, the new valuation $\mathbf{x}'$ is obtained by applying f_τ on $\mathbf{x}$ previously modified according to $\mathbf{x}''_p$. Whenever a p-labelled successor exist in a state, we say that p is *enabled* in that state.

Example 1. *Figure 3 shows three atomic components, Event-Creator,* Event-Receiver1 *and Event-Receiver2. The Event-Creator component contains three control states l_1, l_2 and l_3 and a set of ports $\{crequest, cconfirm, cget, cpush, cancel\}$. Initially at the state l_1, the Event-Creator sends a request to both Event-Receivers by executing the transition labelled with crequest port. If the event participants send back a confirm and transition labelled by $cconfirm$ is executed, both atomic components can exchange data variables $notif$ and $info$ through the Event-Creator, otherwise, the event creation is canceled. The dashed squares represent security annotations that will be presented in the coming sections.*

The system composition is obtained by binding atomic components $\{B_i = (L_i, X_i, P_i, T_i)\}_{i=1,n}$ trough specific composition operators. We consider that atomic components have pairwise disjoint sets of states, 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 $X_i \cap X_j = \emptyset$. We denote $P = \bigcup_{i=1}^{n} P_i$ the set of all the ports, $L = \bigcup_{i=1}^{n} L_i$ the set of all states, and $X = \bigcup_{i=1}^{n} X_i$ the set of all variables.

An *interaction* a between atomic components is a triple (P_a, G_a, F_a), where $P_a \subseteq P$ is a set of ports, G_a is a guard, and F_a is an update 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 $C = \gamma(B_1, \ldots, B_n)$ 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 $B_i = (L_i, X_i, P_i, T_i)$ and $\textsc{lts}(B_i) = (Q_i, \Sigma_i, \xrightarrow[B_i]{})$ their semantics, for all $i = 1, n$. The semantics of C is the labelled transition system $\textsc{lts}(C) = (Q_C, \Sigma_C, \xrightarrow[C]{})$ 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:

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

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 $\textsc{traces}(C, q_0)$.

Example 2. *Figure 3 presents a simplified composite component from the Whens-App application previously presented in Section 2. The composition represents an event creation between two Event-receiver components. Here, interactions are represented using connectors (lines) between the interacting ports. All interactions between components Event-Creator and Event-Receiver are strong synchronized binary interactions. The interactions {get,push} implements a data transfer between both Event-receivers , that is, an assignments at exportation between variables "info" and "notif".*

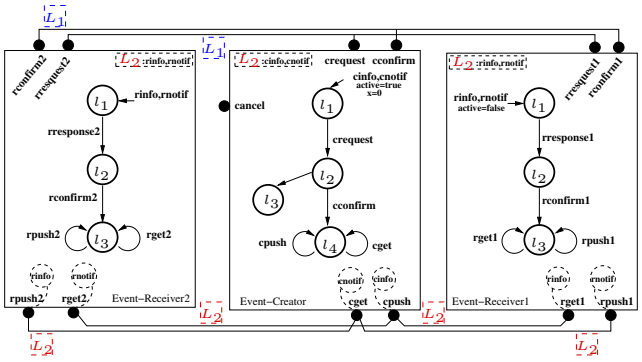

Figure 3: Composite component

3.2 Information Flow Security

We consider information flow policies [8, 4, 11] with focus on the non-interference property. In order to track information we adopt the classification technique and we define a classification policy where we annotate the information by assigning security levels to different parts of the centralized component model (data variables, ports and interactions). The policy describes how information can flow from one classification with respect to the other.

The system parameters are annotated using the *Decentralized Label Model* (DLM) introduced in [13]. In DLM, labels are defined as pair of confidentiality and integrity policies denoted $\{c; d\}$, where c is the confidentiality policy and d is the integrity policy. In the rest of the paper, in order to simplify the notations and since integrity is treated dually, we concentrate only on confidentiality. The main entity used to express policies is the principal. A principal is a atomic

component that has the power to observe and change certain aspects of the system. Principals are ordered using the *can-acts-for* relation ($\preceq$), which is a delegation mechanism that enables a principal to pass some of his rights to another principal.

A confidentiality label L contains an owner set, denoted $O(L)$, that are principals whose data was observed in order to construct the data value; they are the original sources of the information. Label L also 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 value. The association of an owner o and a set of readers $R(o)$ defines a policy. Confidentiality label is expressed using set of policies. For example, considering the confidentiality label L_2 assigned to exported variables $cinfo$,$rinfo$, $cnotif$ and $rnotif$ in figure 3, where L_2 :{*Event-Creator*: *Event-Receiver1*, *Event-Receiver2*} where *Event-Creator* can act for *Event-Receiver1* and *Event-Receiver2*. We denote by $(\_)$ the less restrictive authority, for instance label $L_1 : \{\_ : \_\}$ assigned to the *request* interaction that is considered a public event.

A security domain is a lattice of the form $\langle S, \subseteq, \cup, \cap \rangle$ where:

- S is a finite set of security labels.
- $\subseteq$ is a partial order "can flow to" on S that indicates that information can flow from one security level to an equal or a more restrictive one. For two labels L_1 and L_2, we consider that $L_1 \subseteq L_2$ if and only if $O(L_1) \preceq O(L_2)$ and $\forall o \in O(L_1), R(L_1, o) \preceq R(L_2, o)$.
- $\cup$ is a "join" operator for any two labels in S and that represents the upper bound (LUB) of them. The join of two labels L_1 and L_2 denoted $L_1 \cup L_2$ contains an owner set $O(L_1 \cup L_2) = O(L_1) \cup O(L_2)$ and $\forall o \in O(L_1 \cup L_2)$ there is reader set $R(L_1 \cup L_2, o) = R(L_1, o) \cap R(L_2, o)$
- $\cap$ is a "meet" operator for any two levels in S that represents the lower bound (GLB)of them. The meet of two labels L_1 and L_2 denoted $L_1 \cap L_2$ do contains an owner set $O(L_1 \cup L_2) = O(L_1) \cup O(L_2)$ and $\forall o \in O(L_1 \cup L_2)$ there is reader set$R(L_1 \cup L_2, o) = R(L_1, o) \cup R(L_2, o)$

The intuition behind the definition of $\subseteq$ relation 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 acts 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 by the policy but also any principal able to act for the explicitly mentioned reader is also able to read the data.

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 $\langle S, \subseteq, \cup, \cap \rangle$ be a security domain, fixed.

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

In atomic components, the security levels considered for ports and variables allow to track intra-component information flows and control the intermediate computation steps. Moreover, inter-components communication, that is, interactions with data exchange, are tracked by the security levels assigned to interactions. For example, ports, variables and interactions of previously presented example in Figure 3 are tagged with L_1, L_2 security levels (graphically represented with dashed squares).

We will now formally introduce the notions of non-interference for our component model. We start by providing few additional notations and definitions. Let σ be a security assignment for C, fixed. For a security level $s \in S$, we define $\gamma{\downarrow_s^\sigma}$ the restriction of γ to interactions with security level at most s that is formally, $\gamma{\downarrow_s^\sigma} = \{a \in \gamma \mid \sigma(a) \subseteq s\}$.

For a security level $s \in S$, 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 $\sigma(a) \subseteq s$ and $(aw)|_s^\sigma = w|_s^\sigma$ if $\sigma(a) \not\subseteq s$. 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)*. 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.

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,*

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

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

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 $s \in S$.*

Here non-interference is expressed as 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 L_1 must not be able to distinguish neither changes on variables or occurrence of interactions having higher security level L_2.

3.3 Checking Non-interference

We established sufficient syntactic conditions that aim to simplify the verification of non-interference and reduce it to local constrains check on both transitions (inter-component verification) and interactions (intra-component verification). We recall these conditions in order to be used later in section 4 for rechecking security correctness of the decentralized component model. Indeed, these conditions offer an easy way to automate verification and there preservation proofs the system non-interference.

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:*

$$\forall \tau_1, \tau_2 \in T_i : \ \tau_1 = \ell_1 \xrightarrow{p_1} \ell_2, \ \tau_2 = \ell_2 \xrightarrow{p_2} \ell_3 \Rightarrow \ell_1 \neq \ell_2 \Rightarrow \ \sigma(p_1) \subseteq \sigma(p_2)$$

- *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:

$$\forall y \in use(e) : \ \sigma(y) \subseteq \sigma(x)$$

(iii) variables are consistently used and assigned in transitions and interactions, that is,

$$\forall \tau \in \cup_{i=1}^{n} T_i \forall x, y \in X : x \in def(f_\tau), y \in use(g_\tau) \Rightarrow \sigma(y) \subseteq \sigma(p_\tau) \subseteq \sigma(x)$$
$$\forall a \in \gamma \ \ \forall x, y \in X \ : x \in def(F_a), y \in use(G_a) \Rightarrow \sigma(y) \subseteq \sigma(a) \subseteq \sigma(x)$$

(iv) all atomic components B_i are port deterministic:

$$\forall \tau_1, \tau_2 \in T_i : \ \tau_1 = \ell_1 \xrightarrow{p} \ell_2, \tau_2 = \ell_1 \xrightarrow{p} \ell_3 \Rightarrow (g_{\tau_1} \wedge g_{\tau_2}) \text{ is unsatisfiable}$$

The first family of conditions (i) is similar to Accorsi's conditions [1] for excluding causal and conflicting places for Petri net transitions having different security levels. Similar conditions have been considered in [9, 10] 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 [5], states that the above 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.*

As an illustration, consider the composite component in Figure 3. It can be relatively easily checked that the security conditions hold. Indeed since *Event-Creator* acts for *Event-Receiver1* and *Event-Receiver2*, $L1 \subseteq L_2$ and the security level L_2 of variables at the assignment $cinfo := rinfo$ is consistent with the security level L_1 of the guard's variable *active*. Besides, the security level of ports involved in all interactions is also consistent (equal). Henceforth, the composite component is secure.

4. SECURE DECENTRALIZED MODEL

In this section we first recall the key steps for decentralizing the functional model from Section 3.1 following a transformation method from [6]. This method relies on a systematic transformation of centralized atomic components and replacement of multiparty interaction by protocols expressed using send/receive (S/R) interactions. S/R interactions are binary point-to-point and directed interactions from one sender component (port), to one receiver component (port) implementing message passing. The transformation guarantees that the receive port is always enabled when the corresponding send port becomes enabled, and therefore S/R interactions can be safely implemented using asynchronous message passing primitives (e.g., TCP/IP network communication, MPI communication, etc...). To this end, each atomic component B_i is transformed into a decentralized B_i^{SR} component.

DEFINITION 6 (COMPOSITE S/R COMPONENT). *$C^{SR} = \gamma^{SR}(B_1^{SR}, \ldots, B_n^{SR})$ is a S/R composite component if we can partition the set of ports of B^{SR} into three sets P_s, P_r, P_u that are respectively the set of send-ports, receive-ports and unary interaction ports.*

- Each interaction $a = (P_a, G_a, F_a) \in \gamma^{SR}$ is either (1) a S/R interaction with $P_a = (s, r_1, r_2, ..., r_k)$, $s \in P_s$, $r_1, ..., r_k \in P_r$ and $G_a = true$ and F_a copies the variables exported by the port s to the variables exported by the port $r_1, r_2 ..., r_k$ or (2) a unary interaction $P_a = \{p\}$ with $p \in P_u$, $G_a = true$, F_a is the identity function.
- If s is a port in P_s, then there exists one and only one S/R interaction $a = (P_a, G_a, F_a) \in \gamma^{SR}$ with $P_a = (s, r_1, r_2, ..., r_k)$ and all ports $r_1, r_2, ..., r_k$ are receive ports. We say that $r_1, r_2, ..., r_k$ are the receive ports associated to F.
- If $a = (P_a, G_a, F_a)$ with $P_a = (s, r_1, r_2, ..., r_k)$ is a S/R interaction in γ^{SR} and s is enabled in some global state of C^{SR} then all its associated receive-ports $r_1, r_2, ..., r_k$ are also enabled at that state.

From a functional point of view, the main challenge when transforming functional models with multiparty interactions towards distributed models with send/receive interactions is to enhance parallelism for execution of concurrently enabled interactions and computations within components. That is, in a distributed setting, each atomic component executes independently and thus has to communicate with other components in order to ensure correct execution with respect to the original semantics. The existing method for distributed implementation relies on introducing an interaction protocol layer to handle interactions between decentralized atomic components layer. The first layer (S/R atomic component) includes transformed atomic components. Each atomic component will publish its offer, that is the list of its enabled ports, and then wait for a notification indicating which interaction has been chosen for execution. The second layer (*IP*) deals with distributed execution of interactions by implementing specific interaction protocols. The interaction protocol evaluates the guard of each interaction and executes the associated update function. The interface between this layer and the component layer provides ports for receiving offers and notifying the ports selected for execution.

In the rest of this section, we extend the above decentralization method such that to encompass and preserve information flow security. Here, we impose additional modifications in S/R component behavior to encompass multi-level security annotations, as well as restrictions on the partitioning of the *IP* components. That is, interactions must be statically partitioned according to their security levels to enforce isolation at event level. Despite this partitioning, we show that different security level data still can be managed within a single security interaction manager. Besides, we provide label propagation rules to automatically enforce the non-interference at decentralized model which enforces security at implementation level.

Let $C = \gamma(B_1, \cdots B_n)$ be a composite component and σ be a security assignment for C with domain S, fixed. Moreover, assume that σ satisfies the security conditions defined in subsection 3.3 for C. Furthermore, to simplify presentation, consider that atomic components are deterministic, that is, for every state ℓ and port p there exists at most one transition outgoing ℓ and labelled by p.

4.1 Atomic Components Layer

The transformation at atomic component level consists on breaking the atomicity of there 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 the update function. The intuition behind this transformation is that the offer transition correspond to sending information about component's intention to interact to some scheduler and the notification transition corresponds to receiving the answer from the scheduler, once some interaction has been completed. Update functions can be then executed concurrently and independently by components upon notification reception.

In contrast to [6], to protect the information flow in the distributed context, some changes are needed. A distinct offer port o_s and a different participation number n_s are defined for every security level used within the corresponding atomic component in centralized model. Thus, we ensure that offers and their corresponding notifications have the same security level. Equally, information about execution of interactions having different security levels is not revealed through the observation of n_s variable.

Definition 7 (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* $L^{\perp} = \{\perp_\ell \mid \ell \in L\}$
- $X^{SR} = X \cup \{x_p\}_{p \in P} \cup \{n_s | s \in S\}$ *where each x_p is a Boolean variable indicating whether port p is enabled, and n_s is an integer called a participation number (for security level s).*
- $P^{SR} = P \cup \{o_s \mid s \in S\}$. *The offer ports o_s export the variables $X^{SR}_{o_s} = \{n_s\} \bigcup \{\{x_p\} \cup X_p \mid \sigma(p) = s\}$ that is the participation number n_s, the new Boolean variables x_p and the variables X_p associated to ports p having security level s. For all other ports $p \in P$, we define $X^{SR}_p = X_p$.*
- *For each state $\ell \in L^{SR}$, let S_ℓ be the set of security levels assigned to ports labelling all outgoing transitions of ℓ. For each security level $s \in S_\ell$, we include the following 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} is the identity function.*
- *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. The function f_p applies the original update function f_τ on X, increments n_s and updates the Boolean variables x_r, for all $r \in P$. That is, $x_r := g_\tau$ if $\exists \tau = \ell' \xrightarrow{r} \ell'' \in T$, and false otherwise.*

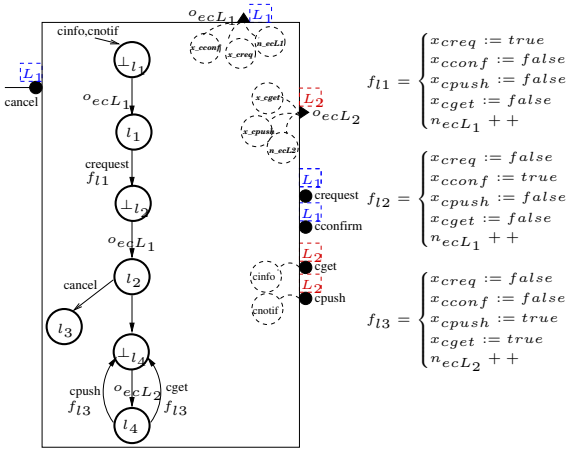

Figure 4: Transformation of the Event-Creator component (Figure 3).

Example 3. *Figure 4 represents the transformed S/R version of the Event-Creator component, presented in Figure 3. The component is initially in control state $\perp_{l_1}$. It sends an offer through the corresponding offer port o_{ecL_1} containing the current enabled port x_{creq} and the participation number n_{ecL_1}, then reaches state l_1. In that state, it waits for a notification from crequest port triggers the execution of the update function which consists on incrementing the value of n_{ecL_1} and re-evaluating x_{creq} and x_{cconf}. At state $\perp_{l_2}$, the Event-Creator sends an offer through port o_{ecL_1} to create an event between participants. The event is created only if all participants are ready to join, otherwise, the event is cancelled.*

Definition 8 (security assignment σ^{SR} for B^{SR}). *The security assignment σ^{SR} is defined as an extension of the original security assignment σ. For variables X^{SR} and ports P^{SR} of a transformed atomic components B^{SR}, define*

$$\sigma^{SR}(x) = \begin{cases} \sigma(p) & \text{if } x = x_p \text{ for some } p \in P \\ s & \text{if } x = n_s \text{ for some } s \in S \\ \sigma(x) & \text{otherwise, for any other } x \in X^{SR} \end{cases}$$

$$\sigma^{SR}(p) = \begin{cases} s & \text{if } p = o_s \text{ for some } s \in S \\ \sigma(p) & \text{otherwise, for any other } p \in P^{SR} \end{cases}$$

As an illustration, reconsider the example depicted in Figure 4. Following the above definition, ports *crequest*, *cconfirm* and o_{ecL_1} are tagged as (L_1) and respectively ports *cpush*, *cget* and o_{ecL_2} are tagged with (L_2) where $L_1 \subseteq L_2$.

Lemma 2. *If the security assignment σ satisfies the security conditions for the atomic component B then the security assignment σ^{SR} satisfies the security conditions for the transformed S/R component B^{SR}.*

4.2 Interaction Protocol (IP) Layer

This layer consists of a set of components, each in charge of execution of a subset of interactions in the initial centralized model. Each component represent a scheduler that receives messages from S/R components then calculates the enabled interaction and selects them for execution. The use of the *IP* components allow parallel execution at components level as well as interactions level in a distributed environment. In this section we show how to construct a secure *IP* schedulers without introducing unexpected behavior nor disallowing interleavings, which represents a compromise between liveness and security property in distributed systems. Indeed, a parallel execution of two distinct security level interaction would not need to suspend one of them to maintain security, thus there will be no internal timing leak.

Conflicts between interactions executed by the same *IP* component are resolved by that component locally. Two interactions a_1 and a_2 are in conflict iff either, they share a common port p (i.e $p \in a_1 \cap a_2$), or there exist two conflicting transitions at a local state ℓ of a component B_i that are labelled with ports p_1 and p_2, where $p_1 \in a_1$ and $p_2 \in a_2$. *IP* components behaviors used in this layer are similar to the ones introduced in [6]. However, to enforce event noninterference we enforce the partitioning of interactions according to there security levels. Let us remark that, for a centralized component model satisfying security conditions, the above partitioning can be proven *conflict-free*, that is, no conflicts exist between interactions having different security levels. Henceforth, all conflicts can be resolved locally by *IP* components. Such a solution is practical and preserve initial system behavior without introducing additional waiting time or priorities at execution to some interactions of certain level over others in schedulers (IP components) to preserve security.

Nonetheless, the security of this centralized solution is not granted for the data part. Remind that in the centralized model, interactions at some security level s are allowed to transfer and/or perform arbitrary computations on data variables with levels other than s. Therefore, in contrast to interactions, annotations of variables requires a bit of care to maintain the security conditions. A concrete example is presented in Figure 6 and discussed later.

Let $C = \gamma(B_1, \cdots, B_n)$ be a composite component and $\gamma_s = \{a_i = (P_i, F_i, G_i) | a_i \in \gamma, \sigma(a_i) = s\}$ the set of interactions of level s. Let *participants*(γ_s) (resp. *ports*(γ_s)) be the set of atomic components (resp. ports) participating (resp. occurring) in interactions from γ_s.

Definition 9 (IP component at level s). *The component $IP_s = (L^{IP}, X^{IP}, P^{IP}, T^{IP})$ handling γ_s is defined as:*

- *Set of places $L^{IP} = \{w_i, r_i \mid B_i \in participants(\gamma_s)\} \cup \{s_p \mid p \in ports(\gamma_s)\}$.*
- *Set of variables $X^{IP} = \{n_{is} \mid B_i \in participants(\gamma_s)\} \cup \{\{x_p\} \cup X_p \mid p \in ports(\gamma_s)\}$*
- *Set of ports $P^{IP} = \{o_{si} \mid B_i \in participants(\gamma_s)\} \cup \{p \mid p \in ports(\gamma_s)\}$ where offer ports o_{is} are associated to variables n_{is}, x_p, and X_p from component B_i and ports p are associated to variables X_p.*
- *Set of transitions $T^{IP} \subseteq 2^{L^{IP}} \times P^{IP} \times 2^{L^{IP}}$. 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 $(\{r_i\}_{i\in I}, a, \{s_{pi}\}_{i\in I})$ for each interaction $a \in \gamma_s$ such that $a = \{p_i\}_{i\in I}$, where I 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 $B_i \in participants(\gamma_s)$.*

Example 4. *Figure 5 illustrates the IP_{L_1} component constructed for interactions $\gamma_{L_1} = \{request, confirm\}$ for the example shown in Figure 3. For all B_i^{SR} components involved in interactions γ_{L_1}, we introduce a waiting (w_i) and receiving (r_i) places (i.e, (w_{er1} ,w_{ec}, w_{er2}) and (r_{er1}, r_{ec}, r_{er2}). For all ports p involved in γ_{L_1} we introduce a sending place s_{p_i} (i.e, (s_{rreq1}, s_{creq}, s_{rreq2}, s_{rconf1}, s_{rconf2} and s_{cconf}). The IP_{L_1} component moves from w_i to rcv_i whenever it receives an offer from the corresponding component B_i^{SR}. After choosing and executing interactions, the IP_{L_1} component moves to sending (s_p) places to send notification through ports p to the corresponding component.*

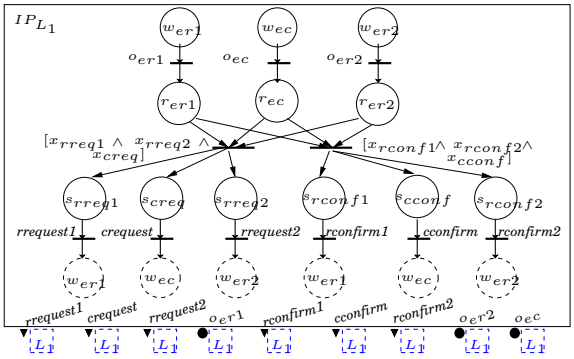

Figure 5: IP_{L_1} Event-secure interactions scheduler component

Definition 10 (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*

$$\sigma^{SR}(x) = \begin{cases} \sigma(x) & \text{if } x \in X_p \text{ and } s \subseteq \sigma(x) \\ s & \text{otherwise} \end{cases}$$

$$\sigma^{SR}(p) = s, \text{ forall } p \in P^{IP}$$

Informally, the security assignment 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 levels at least s. This change is mandatory to ensure consistent copy of data in offers (resp. notifications) from (resp. to) components to the IP.

Example 5. *Figure 6 (a) presents a data transfer between Event-Creator and an Event-Receiver1 on get interaction where the variable $rinfo$ from component Event-Receiver1*

is assigned to the variable cinfo in component Event-Creator. Here we assume that variable rinfo is tagged with L_1 annotation and variables cnotif is tagged with L_2 where $L_1 \subseteq L_2$. In the decentralized model presented in Figure 6 (b), the IP_{L_2} component executes the interaction get. To this end, variable cinfo is imported into a same security level variables cinfo′, while the variable rinfo is imported into a higher security level variable rinfo′, through the corresponding offer port, such that we preserve the security level consistency between interaction (ports) and the transferred variables . Once the interaction takes place, cinfo′ is copied back to cinfo on the notification transition. No copy is performed back to the rinfo, hence, we manage variables of different levels into the same IP scheduler while preserving there initially defined levels.

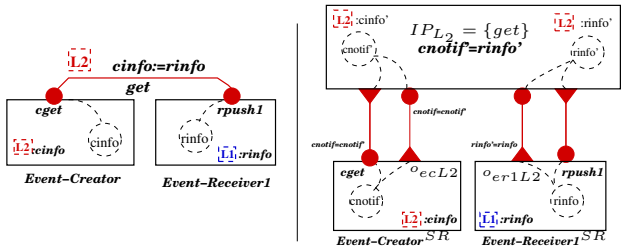

Figure 6: Secure data exchange between atomic and IP components.

LEMMA 3. *IP components satisfies the security conditions with security assignment σ^{SR}.*

Figure 7 represents the distributed model of the system shown in Figure 3 with two distinct security level interaction management. Indeed, low-level security interactions *request* and *confirm* are managed with a single IP_{L_1}, where high level interaction *push* and *get* are managed with the IP_{L_2}.

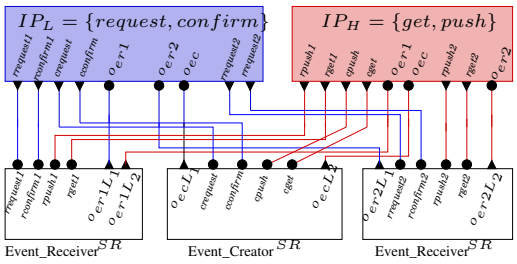

Figure 7: Centralized interaction management

4.3 Cross-layer Interactions

Hereafter, we define the interactions in the S/R composition model Following Definition 6, we introduce S/R interactions by specifying the sender and the associated receivers. Given a composite component $C = \gamma(B_1, \cdots, B_n)$, and partitions $\gamma_{1s}, \cdots, \gamma_{ms}$ of $\gamma_s \subseteq \gamma$, for every security levels $s \in S$, the transformation produces a S/R composite component $C^{SR} = \gamma^{SR}(B_1^{SR}, \cdots, B_n^{SR}$, $(IP_{1s}, \cdots, IP_{ms})_{s \in S})$. We define the S/R interactions of γ^{SR} as follows:

- For every atomic component B_i^{SR} participating in interactions of security level s, for every IP components $IP_{1s}, \cdots, IP_{ms}$ handling γ_s, include in γ^{SR} the *offer* interaction $off_s = (B_i^{SR}.o_s, IP_{1s}.o_{is}, \cdots, IP_{ms}.o_{is})$.
- For every port p in component B_i^{SR} and for every IP_{js} component handling an interaction involving p, we include in γ^{SR} the *response* interaction res_p=$(IP_{js}.p, B_i^{SR}.p)$

DEFINITION 11 (SECURITY ASSIGNMENT σ^{SR} FOR γ^{SR}). *The security assignment σ^{SR} is build from the security assignment σ. For interactions γ^{SR} between all atomic components of the transformed model, we define $\sigma^{SR}(a) = s$ for any interaction a involving an IP_{js} component handling interactions with security level s.*

LEMMA 4. *All the cross-layer interactions of C^{SR} are secure with σ^{SR}.*

The following theorem states the correctness of our transformation, that is, the constructed S/R model satisfies the security conditions by construction.

THEOREM 5 (SECURITY-BY-CONSTRUCTION). *If the component C satisfies security conditions for the security assignment σ then the transformed component C^{SR} satisfies security conditions for the security assignment σ^{SR}.*

PROOF. From lemma 2,3 and 4 we ensure the preservation of all security conditions at all S/R model layer and transformation steps. □

5. IMPLEMENTATION

In this section, we illustrate the complete design flow for generating secure distributed code represented in Figure 8. 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 *BIP* framework [14], we implement these modules in Java language and we generate a C++ code. 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 discuss the different steps and design choices.

5.1 Abstract Model Configuration

Additionally to the system functional model (.bip), the system designer provide a configuration file (Annotations.xml) that contains the DLM annotations from Section 3.2 where we define the acts_for relations and labels to different ports and data in each component. Figure 9 presents fragments of the configuration file for the *Whens-App* abstract model. We extend the system model parser to extract labels from Annotations.xml file. Then, we associate annotations to their corresponding ports and data types stored in the secure component 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

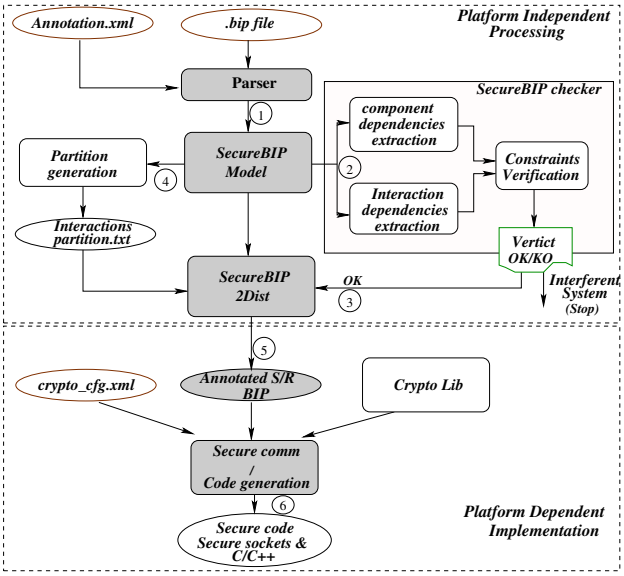

Figure 8: Tool-set Architecture Overview.

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 *IP*component would manage. This file is used as input by *secureBIP 2Dist* to generate an annotated S/R model. The *secureBIP 2Dist* generator is modified to encompass modifications in decentralized model as well as rules for annotations propagation.

```
- <config>
  - <acts_for>
      <authority>Event_Creator:Event_receiver1,Event_Receiver2</authority>
    </acts_for>
  - <var_config>
      <variable name="cinfo" component="Event_Creator"
        label="Event_Creator:Event_receiver1,Event_Receiver2" />
      <variable name="rinfo1" component="Event_receiver1"
        label="Event_Creator:Event_receiver1,Event_Receiver2" />
      <variable name="rinfo2" component="Event_receiver2"
        label="Event_Creator:Event_receiver1,Event_Receiver2" />
      ...
    </var_config>
  - <port_config>
      <port name="crequest" component="Event_Creator" label="_:_" />
      <port name="cconfirm" component="Event_Creator" label="_:_" />
      <port name="cpush" component="Event_Creator"
        label="Event_Creator:Event_receiver1,Event_Receiver2" />
      <port name="cget" component="Event_Creator"
        label="Event_Creator:Event_receiver1,Event_Receiver2" />
      ...
    </port_config>
</config>
```

Figure 9: Configuration file for the abstract model.

5.2 Platform-Dependent Configuration

Here the system designer provides configuration file that contains the cryptographic mechanisms to be used to ensure confidentiality for data and ports to secure interactions between atomic S/R and IP components. To preserve confidentiality we use encryption. We assume that the generated code is running on trusted hosts where it is safe to generate and store encryption keys. The *Crypto Lib* library contains the different encryption protocols and functions that, following the configuration file, the code generator selects messages to secure at communications using secure TCP/IP sockets.

The configuration states the encryption mechanisms for each defined security level, that is, for variables and ports that need to be secured following the secure abstract annotations. A data security is enforced using authentication, encryption and signature mechanisms to encrypt and sign the data at socket buffer before sending it. However we consider that encryption does only provide a degree of privacy with variables. Hence, we enforce the security of ports, if it is configured so, by hiding the message identity at sending to enforce privacy of message sender and receiver.This is done by encapsulating the sent message such that no information can be deduced by observing message transfer between components. In this message source and receiver are encrypted under a shared key between sender and receiver component. The *message_index* (common encrypted pass-world shared between sender and receiver) will be used by the receiver to retrieve the sent message. According to domain application, there exist some privacy extensions allowing the identities of the communicating parties to be hidden from third parties.

Following this defined configuration, we automatically generate stand-alone C++ processes for every S/R components (atomic and *IP*) communicating with secure TCP/IP sockets channels that can be deployed and run on a distributed network. Each C++ process can be run on a host that ensures at least the upper bound security level of annotated data and ports in it. Obviously, it is easier to find a set of hosts that are trusted to run a process of specific security level at most than it is to find a host that can run the whole multi-level system.

5.3 Evaluation

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 and encryption we use OpenSSL library which contains tested C libraries and here we use X.509 certificates for signature and an asymmetric encryption algorithm (RSA) with 2048bit key size.

Components	Compilation (s)	Execution(s)	
		$\gamma(B^{SR})$	Sec-$\gamma(B^{SR})$
3	3.02	4.6	7.1
11	3.84	8.6	10.3
25	4.15	12.5	15.7
101	4.91	22.1	25.2

Table 1: Whens-App configuration and execution time (in s)

As an Evaluation of our approach performance, Table 1 presents some experiments over compilation time that include security check and model transformation and the execution time of the generated code for different component number of *Event-Receivers* in the *Whens-App* system with the use of security mechanisms (*Sec-*$\gamma(B^{SR})$) and without ($\gamma(B^{SR})$). The number of decentralized multi-party interactions for a system with 100 components is 50. the number of binary interactions executed in the decentralized model has reached 480 interaction for a system of 100 components managed with two IP components. Here we can see, that there is no significant overhead at compilation time with the increase of system component. The use of cryptographic mechanisms induces an overhead of 20%, however this per-

formance can be improved if we choose to use,for instance, symmetric encryption instead of the asymmetric one currently implemented. Despite that our prototype implementation is based on the use of secure TCP/IP sockets, the interaction protocol implementation can be flexible and encompass other communication types depending on application domain, such as RMI.

6. RELATED-WORK

Previous works are mainly related to model-based security and deals wit both event and data IFC and the multi-party security.

Model-based security: Several works on model-based security aim at simplifying security configuration and coding [7, 3]. In [3], authors, propose modeling security policy in UML and target automating security code generation for business applications like JEE and .net applications. Other works [7] 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. Recent works on information flow security in web services rely on Petri-nets for modeling composed services [2]. Petri-net graphs are generated from *BPEL* orchestration processes and are, next, modified by the developer to represent shared resources and to annotate interactions. Developer's modification is necessary here since Petri-nets capture event-based interactions only. Our model allows representing both data and events. Furthermore, in [2], proposed tools allowing only non-interference checking and no security enforcement is proposed.

Multi-party security: This domain deals with "data-mining privacy preservation", [12]. The goal is that a set of parties with private inputs wishes to jointly compute some function of their inputs. The parties learn the correct output and nothing else. A typical application is the one that need to publicize tables that sum up some statistics. This task is extremely dangerous because census questionnaires contain a lot of sensitive information, and it is crucial that it not be possible to identify a single user in the publicized tables. Compared to these works that generally deal with low level cryptography protocols verification, we provide a more abstract model that helps system designers to check their system security more rapidly. Indeed, it is very important to check security configuration early in the life cycle of system development. Before detailing the communication protocols and fine-grained system coding, the designer checks security configurations starting from a high level description of component's behavior and interactions.

7. CONCLUSION

In this paper, we present a practical approach to automatically secure information flow in distributed systems. Starting from an abstract component-based model with multi-party interactions, we verify security policy preservation as defined by the user, that is, verifying non-interference property at both event and data levels. Then, we generate a distributed model where multiparty interactions are replaced with asynchronous message passing. The intermediate distributed model is formally proved "secure-by-construction". Here we provide a framework with a set of tools allowing to build distributed systems and automatically generating secure code that implements the desired security policy defined at the centralized level. This work is now being extended to verify this approach on a parametric model with dynamic labelling annotations. We are also trying to apply our decentralization method to a relaxed version of non-interference (e.g, intransitive or with declassification mechanisms), since for some systems, the transitive definition of non-interference is relatively strict for practical use.

8. REFERENCES

[1] Accorsi, R., Lehmann, A.: Automatic information flow analysis of business process models. In: Proceedings of the 10th international conference on Business Process Management, BPM'12 (2012)

[2] 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)

[3] Basin, D., Doser, J., Lodderstedt, T.: Model driven security: from uml models to access control infrastructures. ACM Transactions on Software Engineering and Methodology p. 2006

[4] Bell, E.D., La Padula, J.L.: Secure computer system: Unified exposition and multics interpretation (1976)

[5] Ben Said, N., Abdellatif, T., Bensalem, S., Bozga, M.: Model-driven information flow security for component-based systems. In: Proceedings of Etaps workshop 2014, From Programs to Systems,The Systems Perspective in Computing (2014)

[6] Borzoo, B., Marius, B., Mohamad, J., Jean, Q., Joseph, S.: A framework for automated distributed implementation of component-based models. Distributed Computing (2012)

[7] Chollet, S., Lalanda, P.: Security Specification at Process Level. 2008 IEEE International Conference on Services Computing (2008)

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

[9] Focardi, R., Rossi, S., Sabelfeld, A.: Bridging language-based and process calculi security. In: In Proc. of Foundations of Software Science and Computation Structures (FOSSACS'05), volume 3441 of LNCS, pp. 299–315. Springer-Verlag (2005)

[10] Frau, S., Gorrieri, R., Ferigato, C.: Formal aspects in security and trust (2009)

[11] Goguen, J., Meseguer, J.: Security policies and security models. In: Proceedings of the 1982 ieee symposium on security and privacy, pp. 11–20. IEEE Computer Society

[12] Lindell, Y., Pinkas, B.: Secure multiparty computation for privacy-preserving data mining. IACR Cryptology ePrint Archive (2008)

[13] Myers, A.C., Liskov, B.: Protecting privacy using the decentralized label model. ACM Trans. Softw. Eng. Methodol. (2000)

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