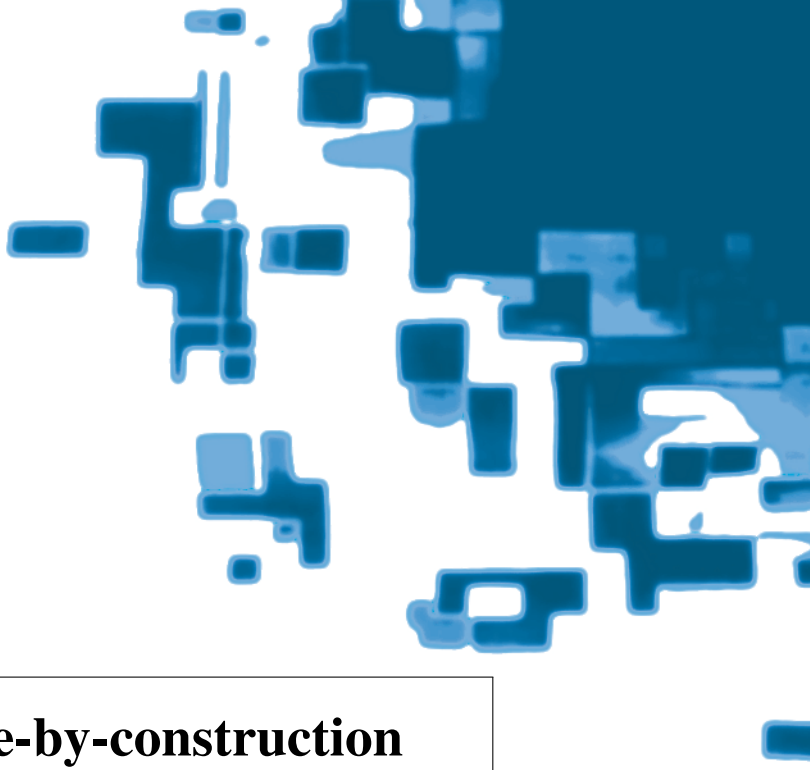

Building secure-by-construction distributed component-based systems

Najah Ben Said , Takoua Abdellatif, Saddek Bensalem, Marius Bozga

Verimag Research Report n^o TR-2014-6

April 2014

Reports are downloadable at the following address
http://www-verimag.imag.fr

Unité Mixte de Recherche 5104 CNRS - Grenoble INP - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

Building secure-by-construction distributed component-based systems

Najah Ben Said , Takoua Abdellatif, Saddek Bensalem, Marius Bozga

April 2014

Abstract

We present an automated method to build secure distributed systems from an abstract multi-level security component-based model. We take as input a high-level *secureBIP* component-based model and transform it into a decentralized Send/Receive *secureBIP* model and further on distributed code. The security policy is defined at the design time. Information flow policy is verified and automatically preserved on intermediate models towards distributed implementation. The distributed implementation is therefore proven "secure-by-construction" that is, the executable code conforms to the desired security policy. The method has been implemented and we present experimental results obtained on a case study.

Keywords: component-based distributed systems, information flow security, non-interference, model transformation, model-based security.

Reviewers:

Notes: The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement no. 318772 - Distributed MILS for Dependable Information and Communication Infrastructures (D-MILS).

How to cite this report:

```
@techreport {TR-2014-6,
  title = {Building secure-by-construction distributed component-based systems},
  author = {Najah Ben Said , Takoua Abdellatif, Saddek Bensalem, Marius Bozga},
  institution = {{Verimag} Research Report},
  number = {TR-2014-6},
  year = {}
}
```

1 Introduction

Preserving information flow security is significantly challenging in the context of distributed systems. Exchanging information across multiple, public or possibly untrusted networks increases information's vulnerability to attacks by mischievous elements. The security of distributed systems usually entails the implementation of cryptographic primitives and access control techniques which are essential tools to ensure confidentiality and integrity of information flow. Although they are widely used, access control techniques are rather limited since by only providing capacities to different users to read or modify information there is no guarantee of the security of information propagation in the system especially with non-trivial interactions. Encryption primitives are also less helpful to ensure that the system obeys an overall security policy.
We are interested on ensuring information flow security by verifying and enforcing non-interference property [13]. We build a component-based framework, *secureBIP*, which allows building secure, complex and hierarchically structured systems by describing atomic components behavior and interactions. *SecureBIP* formally defines two types of non-interference: event-based non-interference [18, 17, 21, 19] and data non-interference generally addressed in security-typed programming languages [20]. Considering both non-interference models allows addressing a larger range of security attacks and implementing a fine-grained information flow security compared to solutions addressing a unique model. Moreover,we identified a set of sufficient syntactic conditions allowing to automate verification of non-interference.

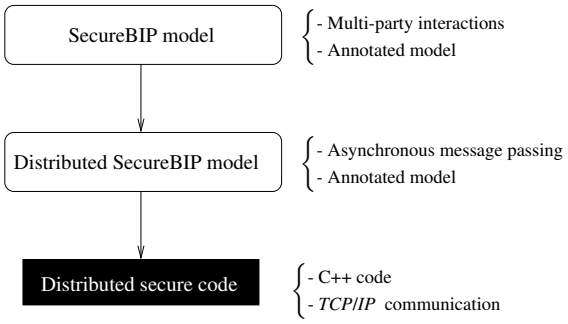

Figure 1: Model Transformation

Starting from a high-level centralized model, the system is described as a set of components and interactions. Annotations at the level of component interfaces and interactions allow to configure the system security policies. Non-interference property is checked to verify that no information can be leaked in an explicit or implicit way. The centralized model is transformed to a decentralized send/receive S/R *secureBIP* model that is, scheduling and communication protocol components are inserted to solve conflicting interactions and set-up distributed communication protocols. Security annotations are translated to the new distributed component-based architecture while preserving information flow security 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 [7], non-interference is preserved on the intermediate S/R *secureBIP* model. 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 communication primitives with security guarantees.
Our contribution can be summarized in the following points:

- We define an automated method based on model transformations to build a secure-by-construction 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 the proofs of the model transformation's correctness 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 security in distributed systems.

- We present a framework that implements our method and use it to secure a Web-Service application.

This paper is based on previous work on non-interference for centralized systems [7]. The paper is structured as follows. Section 2 presents the main concepts of the component-based framework adopted in this work as well as non-interference definitions and the sufficient conditions. In section 3, we describe the automated distribution approach to derive secure executable code. Next, in section 4 we provide a use-case as illustrative example and we discuss implementation and experimental results obtained. Section 5 discusses the related work and section 6 concludes and presents some perspectives for future work. All the proofs of technical results are given in the appendix.

2 Secure Component-Based Model

The *secureBIP* [7] framework is an extension of the *BIP* component framework [5, 4] with security features. We recall the main concepts behind *secureBIP* with a particular focus on security annotations and the different notions of non-interference and their verification.

2.1 The BIP Framework

BIP - Behavior, Interaction, Priority - [5, 4] proposes a layered component architecture model. At lower layer, the behavior is expressed as a set of atomic components, that is, finite state automata or 1-safe Petri nets, extended with data. The middle layer, interactions express synchronization constraints and do the transfer of data between the interacting components. The third layer, priorities are used to filter amongst possible interactions and to control system evolution. In the following, we recall the key concepts of *BIP* 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. Priorities are not considered in this work.

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 $\underset{B}{\longrightarrow}$ is defined by the rule:

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

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 2 shows an atomic component that contains two control states l_2 and l_3 and three ports p_3, p_4 and p_5. Initially at state l_2, the transition labelled by p_4 can only occur if the transition labelled by*

p_3 was executed once, at least, and the variable x is incremented. The exported variable x is associated to the port p_5. The dashed squares represent security annotations and will be presented in the coming sections.

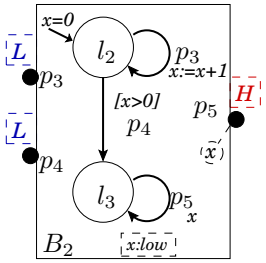

Figure 2: Atomic component

Composite components are obtained by composing an existing set of 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 $\text{LTS}(B_i) = (Q_i, \Sigma_i, \xrightarrow[B_i]{})$ their semantics, for all $i = 1, n$. The semantics of C is the labelled transition system $\text{LTS}(C) = (Q_C, \Sigma_C, \xrightarrow[C]{})$ 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:

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

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

Example 2 *In a composite component, as depicted in Figure 3, interactions are represented using connectors (lines) between the interacting ports. All interactions between components B_1, B_2 and B_3 are strong synchronized binary interactions. The c interaction implements a data transfer between components B_2 and B_3, that is, the update function assigns the variable x (of B_2) to variable z (of B_3).*

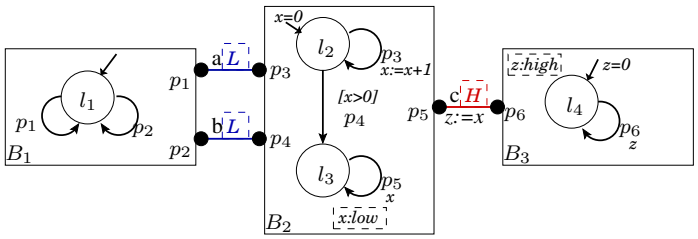

Figure 3: Composite component

2.2 Information Flow Security

We explore information flow policies [9, 6, 13] 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 *secureBIP* model (data variables, ports and interactions). The policy describes how information can flow from one classification with respect to the other.
As an example, we can classify public information as a Low (L) security level and secret (confidential) information as High (H) security level. Intuitively High security level is more restrictive than Low security level and we denote it by $L \subseteq H$. In general, security levels are elements of a security domain, defined as follows:

Definition 3 (security domain) *A security domain is a lattice of the form $\langle S, \subseteq, \cup, \cap \rangle$ where:*

- *S is a finite set of security levels.*
- *$\subseteq$ 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.*
- *$\cup$ is a "join" operator for any two levels in S and that represents the upper bound of them.*
- *$\cap$ is a "meet" operator for any two levels in S and that represents the lower bound of them.*

In the example above, the set of security levels is $S = \{L, H\}$ and the "can flow to" partial order relation is defined as $L \subseteq L, L \subseteq H, H \subseteq H$.
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 4 (security assignment σ) *A security assignment for component C is a mapping $\sigma : X \cup P \cup \gamma \rightarrow S$ 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 examples in Figure 2 and Figure 3 are tagged with High (H) and Low (L) 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 notions of non-interference.

Definition 5 (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,*

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

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

Event non-interference (*ENI*) ensures isolation/security at interaction level. The definition excludes the possibility to gain any relevant information about the occurrences of interactions (events) with strictly greater (or incomparable) levels than s, from the exclusive observation of occurrences of interactions with levels lower or equal to s. That is, an external observer is not able to distinguish between the case where such higher interactions are not observable on execution traces and the case these interactions have been actually statically removed from the composition. This definition is very close to Rushby's [15] definition for transitive non-interference. But, let us remark that event non-interference is not concerned about the protection of data.
Data non-interference (*DNI*) provides isolation/security at data level. The definition ensures that, all states reached from initially indistinguishable states at security level s, by execution of arbitrary but identical traces whenever projected at level s, are also indistinguishable at level s. That means that observation of all variables and interactions with level s or lower excludes any gain of relevant information about variables at higher (or incomparable) level than s. Compared to event non-interference, data non-interference is a stronger property that considers the system's global states (local states and valuation of variables) and focus on their equivalence along identical execution traces (at some security level).
In the *secureBIP* framework, a security assignment 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 $s \in S$.

2.3 Checking Non-interference

We provide hereafter 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). Especially, they give an easy way to automate the verification.

Definition 6 (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:

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

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 [10, 12] 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 [7] states that the above security conditions are sufficient to ensure both event and data non-interference.

Theorem 1 (Corollary 1 of [7]) *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, henceforth, the composite component is secure.

3 Secure Decentralized Model

In this section we describe the decentralization method for *secureBIP* and provide formal proofs for security preservation. This method relies on a systematic transformation from arbitrary *BIP* components into distributed *BIP* components with S/R interactions previously introduced in [8]. 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 any asynchronous message passing primitives (e.g., TCP/IP network communication, MPI communication, etc...).

Definition 7 (composite S/R component) *$C^{SR} = \gamma^{SR}(B_1^{SR}, \ldots, B_n^{SR})$ is a S/R secureBIP 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.*

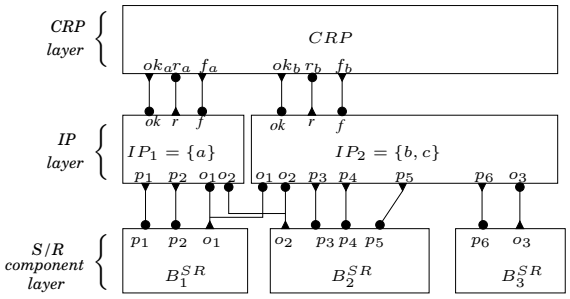

Figure 4: 3-Layer Architecture

- 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 a *BIP* component into a distributed S/R *BIP* component is to enhance parallelism for execution of concurrent interactions. 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 of *BIP* relies on structuring the distributed components according to a hierarchical architecture with three layers, as depicted in Figure 4:

- 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.
- The third layer (*CRP*) deals with conflict resolution between *IPs*. A conflict occurs if two different *IP* components try to execute two interactions involving a common atomic component in the lower layer. Several distributed algorithms exist for conflict resolution, this layer is generic with appropriate interfaces.

The existing method in [8] has been designed without considering any security concerns. In the following, we will show that it can be adapted 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 and conflict resolution layers. In this case, we show that the the security assignment from the original *secureBIP* component is naturally lifted to the distributed component and non-interference is preserved along the transformation.
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 2.3 for C. Furthermore, to simplify presentation, consider that atomic components are deterministic, that is, for every state ℓ and for every port p there exists at most one transition outgoing ℓ which is labelled by p.

3.1 Atomic Components Layer

Referring to the [8] transformation method, the atomicity of transitions is broken at the atomic component level. 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.

Definition 8 (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, for all $r \in P$:*

$$x_r := \begin{cases} g_\tau & \text{if } \exists \tau = \ell' \xrightarrow{r} \ell'' \in T \\ false & \text{otherwise} \end{cases}$$

Example 3 *Figure 5 represents the transformed S/R version of the atomic component presented in Figure 2. The component is initially in control state $\perp_{l2}$. It sends an offer through the corresponding offer port o_{2l} containing the current enabled ports x_{p3}, x_{p4} and the participation number n_{2l}, then reaches state l_2. In that state, it waits for a notification on either port p_3 or p_4. The notification on p_3 triggers the execution of the update function which consists on incrementing the variable x, incrementing the value of n_{2l} and re-evaluating x_{p_3} and x_{p_4} based on the guards of transition labelled with p_3 and p_4 from l_2).*

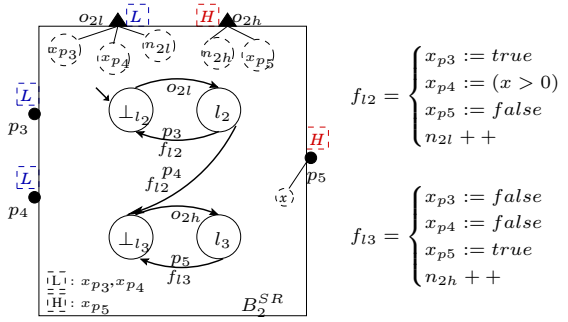

Figure 5: Transformation of atomic components

Definition 9 (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 5. Following the above definition, ports p_3, p_4 and o_{2l} are tagged as Low (L) and respectively ports p_5, o_{2h} are tagged as High (H).

Lemma 1 *If the security assignment σ satisfies the security conditions for the atomic component B then the security assignment σ^{SR} is satisfies the security conditions for the transformed S/R component B^{SR}.*

Proof: We show that transformed S/R atomic components are secure by construction, that is, security conditions (*i*), (*ii*), (*iii*) and (*iv*), related to events and data are preserved by the transformation.

- Condition (*i*): In a transformed atomic B^{SR} component we distinguish two cases of conflicting transitions:

  1. $\tau_1 = \bot_\ell \xrightarrow{o_s} \ell$ and $\tau_2 = \bot_\ell \xrightarrow{o_{s'}} \ell$
  2. $\tau_1 = \ell \xrightarrow{p_1} \bot_{\ell_1}$ and $\tau_2 = \ell \xrightarrow{p_2} \bot_{\ell_2}$.

  From hypothesis, B annotated by σ satisfies security conditions. Hence, from condition (*i*) related to conflicting transitions in B the case (1) can not take place since ports labelling outgoing transitions of state ℓ have the same security level and, moreover, in case (2), it implies that $\sigma^{SR}(p_1) = \sigma^{SR}(p_2)$.

  Similarly, in the transformed component there are two cases of causal transitions:

  1. $\tau_1 = \bot_\ell \xrightarrow{o_s} \ell$ and $\tau_2 = \ell \xrightarrow{p} \bot_{\ell_1}$ and
  2. $\tau_1 = \ell \xrightarrow{p} \bot_{\ell_1}$ and $\tau_2 = \bot_{\ell_1} \xrightarrow{o_{s'}} \ell_1$

  By construction, in (1) $\sigma^{SR}(o_s) = \sigma(p)$. Hence, condition related to causal transitions is verified and $\sigma^{SR}(o_s) \subseteq \sigma(p)$. In (2) $\sigma^{SR}(o_{s'}) = \sigma^{SR}(p')$ such that p' belong to the set of ports labelling outgoing transitions of ℓ'. By assumption, initial atomic component B satisfies security conditions, thus $\sigma(p) \subseteq \sigma(p')$ and by construction $\sigma^{SR}(p) \subseteq \sigma^{SR}(p')$. Therefore $\sigma^{SR}(p) \subseteq \sigma^{SR}(o_{s'})$ which satisfies security conditions (*i*).

- Condition (*ii, iii*): we verify the security level consistency of variables assigned in transitions. All actions defined on transitions of atomic component B are kept unchanged in B^{SR} and the security level of all variables are preserved with σ^{SR}. Hence, by construction these actions are still secure and satisfy conditions (*ii*) and (*iii*). The x_{p_i} variables of enables ports p_i on a state $\{\bot_{\ell_i}\}_{\ell_i \in L^{SR}}$ are modified at the received notification transition labelled with port p at the same state $\bot_{\ell_i}$ where (1) $\sigma^{SR}(x_{p_i}) = \sigma^{SR}(p_i)$. From condition (*i*), we have security level consistency of causal transition, thus (2) $\sigma^{SR}(p) \subseteq \sigma^{SR}(p_i)$. Each variable x_{p_i} is evaluated according to the guard $g_{\tau_{p_i}}$. For all $y \in g_{\tau_{p_i}}$ we have (3) $\sigma^{SR}(y) \subseteq \sigma^{SR}.(p_i)$. From (1), (2) and (3) we can deduce that the condition (*iii*) is preserved for all x_{p_i} variables. The participation number n_s is only incremented with notification transitions labelled with port p having the same security level s. Thus $\sigma^{SR}(p) \subseteq \sigma^{SR}(n_s)$, condition (*ii*), is valid in all transitions.

- condition (*iv*): This condition is trivially verified whenever the atomic component B is deterministic where, for every state there is at most one transition that is labelled by each port.

□

3.2 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 *secureBIP* model. Each component represent a scheduler that receives messages from S/R components then calculates the enabled interaction and selects them for execution.
IP components cannot however run totally independent because of conflicting interactions. 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 $p_2 \in a_2$. Conflicts between interactions executed by the same *IP* component are resolved by that component locally. In contrast, conflicts between interactions executed by different *IP* components are resolved with the help of the conflict resolution layer.
For the case of *secureBIP* systems we present hereafter two configurations of the *IP* layer. The first, called *centralized management* uses the partitioning of interactions following their security level. If the original system satisfies the security conditions, it can be proven that the partitioning is conflict-free, that is, all conflicts are local and can be therefore solved locally by *IP* components. The second solution, called *decentralized management* uses any finer partitioning. That is, the only restriction is that all the interactions handled by some *IP* component have the same security level. In this case, conflicts may exists but *only* between *IP* components handling interactions of the same security level. In this case, the additional conflict resolution layer will be used.

3.2.1 Centralized interactions management

Interaction partitioning is automatically enforced by security types. With a centralized management, all interactions having the same security level are handled by a distinct *IP* component. 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 10 (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 $L^{IP} = \{w_i, rcv_i \mid B_i \in$ participants$(\gamma_s)\} \cup \{snd_p \mid p \in$ ports$(\gamma_s)\}$.*
- *Set of variables $X^{IP} = \{n_{is} \mid B_i \in$ participants$(\gamma_s)\} \bigcup \{\{x_p\} \cup X_p \mid p \in$ 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}, rcv_i) for all components $B_i \in$ participants(γ_s).*
  - *executing interaction $(\{rcv_i\}_{i\in I}, a, \{snd_{pi}\}_{i\in I})$ 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 (snd_p, p, w_i) for all ports p and component $B_i \in$ participants(γ_s).*

Example 4 *Figure 6 illustrates the IP_L component constructed for the L level security interactions $\gamma_L = \{p_1p_3, p_2p_4\}$ for the example shown in Figure 3. For all B_i components involved in interactions γ_L, we introduce a waiting (w_i) and receiving (rcv_i) places (i.e, (w_1, w_2) and (rcv_1, rcv_2). For all ports p involved in γ_L we introduce a sending place snd_p (i.e, $(snd_{p_1}, snd_{p_2}, snd_{p_3}, snd_{p_4})$. The IP_L component moves from w_i to rcv_i whenever it receives an offer from the corresponding component B_i. After choosing and executing interactions, the IP_L component moves to sending (snd_p) places to send notification through ports p to the corresponding component.*

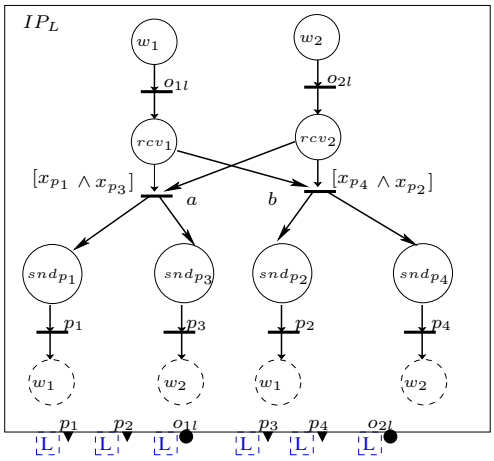

Figure 6: IP_L component managing low interactions

Definition 11 (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*

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

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

Within IP_s component, all ports are annotated with security level s. The imported variables from atomic component B_i having security level higher or equal to s are copied into variables annotated with the same security level $\sigma(x)$. Whereas, imported variables with security level lower or incomparable to s are copied into variables having s as security level. This definition ensures the preservation of security conditions within *IP*component components.

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

Proof: We verify the security of *IP*$_s$ component handling γ_s interactions at event and data level. Security level consistency of causal and conflicting transitions can be easily verified in the *IP*$_s$ components. Indeed, all ports labeling all transitions of the *IP*$_s$ have the same security level s, therefore, the security condition (i) related to causal and conflicting transitions is satisfied at *IP*$_s$ by construction.
Next, we check the security level consistency of data at different transitions of IP_s component: At execution transition, we consider an interaction $a = (G_a, P_a, F_a) \in \gamma$. By assumption, the system initially satisfies security conditions (iii) at interaction level, that is $\sigma(y) \subseteq \sigma(a) \subseteq \sigma(x)$ for all $y \in use(G_a)$ and $x \in def(F_a)$. At the IP component we use copies of variables of the interaction a, where for all $y \in use(G_a)$ there exist a y' such that $y' = y$ and for all $x \in def(F_a)$ there exist a x' such that $x' = x$. By transformation, $\sigma^{SR}(y') = s$, $\sigma^{SR}(x') = \sigma(x)$ and for all $p_i \in \{P_a\}$, $\sigma^{SR}(p_i) = \sigma(a) = s$. With σ^{SR} the condition (iii), $\sigma^{SR}(y') \subseteq \sigma^{SR}(p_i) \subseteq \sigma^{SR}(x')$, is preserved. □
Figure 7 represents the distributed model of the system shown in Figure 3 with centralized interaction management. Indeed, low-level security interactions a and b are managed with a single *IP*$_L$, where high level interaction c is managed with a centralized *IP*$_H$ component.

3.2.2 Decentralized interactions management

To improve the system performance, one can choose to split the set of same security level interactions γ_s into distinct subsets of interactions γ_{js} that are executed by different IP_{js} components. Hence, conflicts can occur between interactions handled by distinct IP_{js} components, e.g in Figure 8 interactions a and b are in conflict at state l_2 of component B_2^{SR}. To solve this conflict the *IP layer* has to communicate with

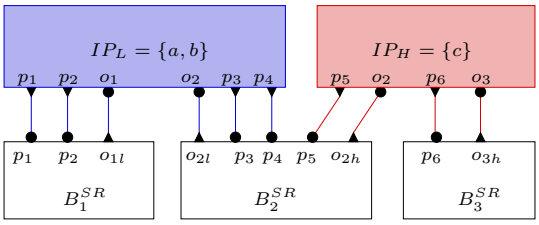

Figure 7: Centralized interaction management

the upper layer to take decisions and select the interaction to execute. To this purpose and for all $a \in \gamma_{is}$ iff $\exists a' \in \gamma_{ks}$ where $k \neq j$ and a is conflicting with a' we include to the previous definition of the IP_s component the following:

- Place: $\{t_a\}$ the *try* places.
- Ports: $\{r_a, ok_a, f_a\}$ where we associate to port r_a a set of participation numbers $\{n_{is}\}_{i \in I}$ where I is the set of all B_i components involved in interaction a.
- Transitions: (1) reservation request $(\{rcv_i\}_{i \in I}, r_a, \{t_a\})$ guarded with $\bigwedge_{p \in a} x_{pi} \wedge G_a$, (2) positive CRP_s response $(\{t_a\}, ok_a, \{snd_{pi}\}_{i \in I})$ and (3) negative CRP_s response $(\{t_a\}, f_a, \{rcv_i\}_{i \in I})$.

The security level of added ports to the decentralized version of IP_{js} are equal to s, that is, $\sigma^{SR}(r_a) = \sigma^{SR}(ok_a) = \sigma^{SR}(f_a) = s$. Thus, we have consistency of the security level of causal and conflicting transitions (condition(i)) and IP_{js} components are secure.
Figure 8 represents the distributed model of system shown in Figure 3 with decentralized interaction management. The low-level security interactions a and b are now managed by IP_{1L} and IP_{2L}, respectively. In this case, the conflict between both interactions a and b which requires the use of CRP_L component to coordinate between IP_{1L} and IP_{2L} components and to schedule the interactions execution.

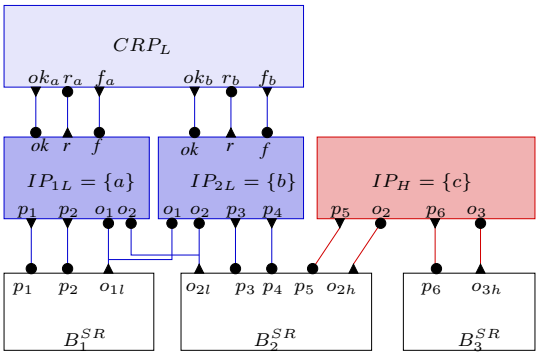

Figure 8: Decentralized interaction management

3.3 Conflict Resolution (CRP) Layer

This layer implements a distributed algorithm that solves the committee coordination problem [16]. Conflicts between interactions of the same security level s hosted by distinct IP_{js} components are managed by a centralized CRP_s component. That is, to preserve non-interference we use a distinct conflict resolution component for each security level. As explained earlier, this is possible only because whenever the original *secureBIP* system satisfies the security conditions, conflicts can occur only between interactions having the same security level.

For the time being, we restrict ourselves to a centralized model of CRP_s as a single atomic component. In this case, for each component B_i participating to interactions having security level s, the CRP_s maintains a participation number N_{is} that stores the latest used value of the participation number n_{is} of component B_i. Its behavior is as follows. Whenever a *reservation* message r_a for interaction $a = \{p_i\}_{i\in I}$ is received by CRP_s, the message provides a set of participation numbers ($\{n_i s\}_{i\in I}$) for all components involved in a. If for each component B_i, the participation number n_{is} is greater than N_{is}, then the CRP_s acknowledges successful reservation through port ok_a and the participation numbers in the CRP_s are set to values sent by the IP_s. On the contrary, if there exists a component whose participation number is less than or equal to what the CRP_s has recorded, then the corresponding component has already participated for this number and the *CRP* replies failure via port f_a.
Let us notice that ports and variables in each CRP_s component have the same security level s, and therefore they trivially satisfy the security conditions (by construction). Thus, CRP_s are secure for all $s \in S$.

Example 5 *Figure 9 illustrate the CRP_L component needed for conflict resolution in the example presented in Figure 8.*

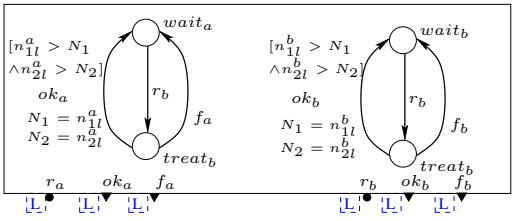

Figure 9: Centralized model of CRP_L

3.4 Cross-layer interactions

In this subsection, we define the interactions in the 3-layer model. Following Definition 7, 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 $C^{SR} = \gamma^{SR}(B_1^{SR}, \cdots, B_n^{SR}, (IP_{1s}, \cdots, IP_{ms})_{s\in S}, \{CRP_s\}_{s\in S})$. 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=($B_i^{SR}.o_s$, $IP_{1s}.o_{is}$,..., $IP_{ms}.o_{is}$).

- 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=($IP_{js}.p$, $B_i^{SR}.p$)

- For every IP_{js} component handling an interaction a that is in conflict with an other interaction a' handled by some different *IP*, include in γ^{SR} the *reserve* interaction r=($IP_{js}.r_a$, $CRP_s.r_a$). Likewise, include in γ^{SR} the *ok* interaction ok=($CRP_s.ok_a$, $IP_{js}.ok_a$) and the *fail* interaction f=($CRP_s.f_a$, $IP_{js}.f_a$).

Definition 12 (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 3 *All the cross-layer interactions of C^{SR} are secure with σ^{SR}.*

Proof: We verify the security level consistency of transferred data on different interactions:

- At *offer* interaction off_s, we perform a copy of received variables through offer ports from B_i^{SR} component to the IP component, such that $\forall x \in \{\{X_p\}_{p\in P} \cup \{x_p\}_{p\in P} \cup \{n_{si} | B_i \in participants(\gamma_s), s \in S\}\}$ there exist a $x' \in X^{IP}$ where $x' := x$. By transformation, $\sigma^{SR}(x') = s$ if $\sigma(x) \subseteq s$ and $\sigma^{SR}(x') = \sigma(x)$ otherwise. The security level of updated variables (x') and used variables (x) are consistent with security level of there corresponding offer interaction, where $\sigma^{SR}(off_s) = s$. Thus, the security condition (ii) and (iii) are preserved at offer interactions.

- At *response* interactions res_p, we send notifications to the corresponding ports p associated with the updated variables x', where $\sigma^{SR}(p) \subseteq \sigma^{SR}(x')$. By construction, $\sigma(p) = \sigma^{SR}(p) = \sigma^{SR}(res_p)$, thus, $\sigma^{SR}(res_p) = \sigma^{SR}(x')$ which satisfies condition (iii).

- With interactions r, f and ok, the only exchanged variables are the participation numbers n_{si} of each participating component i. By construction, each CRP_s and IP_s components are handling interactions of the same security level s, thus $\sigma^{SR}(r) = \sigma^{SR}(f) = \sigma^{SR}(ok) = \sigma^{SR}(n_{si})$. Therefore the security condition (iii) is preserved at these interactions.

□

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

Theorem 2 (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 1,2 and 3 we ensure the preservation of all security conditions at all S/R model layer and transformation steps. □

4 Implementation and Experiments

We illustrate the secure decentralization method with a typical example, a simplified web service reservation system introduced in [7]. A businessman, living in France, plans to go to Berlin for a private and secret mission. To organize his travel, he uses an intelligent web service who contacts two travel agencies: The first agency, *AgencyA*, arranges flights in Europe and the second agency, *AgencyB*, arranges flights exclusively to Germany. The reservation service obtains in return specific flight information and their corresponding prices and chooses the flight that is more convenient for him.
In this example, there are two types of interference that can occur, (1) data-interference since learning the flight price may reveal the flight destination and (2) event interference, since observing the interaction with *AgencyB* can reveal the destination as well. Thus, to keep the mission private, the flight prices and interactions with *AgencyB* have to be kept confidential.
The modeling of the system using *secureBIP* involves two main distinct steps: first, functional requirements modeling reflecting the system behavior, and second, security annotations enforcing the desired security policy. The model of the system has four components denoted: *TravelA* and *TravelB* who are instances from the same component and correspond respectively to *AgencyA* and *AgencyB*, and components *Reservation* and *Payment*. To avoid Figure 10 cluttering, we did not represent the interactions with *TravelA* component. Search parameters are supplied by the user through the *Reservation* component ports *dests* and *dates* to which we associate respectively variables ($from$, to) and $dates$. Next, through search interaction, *Reservation* component contacts *TravelB* component to search for available flights and obtains in return a list L of specific flights with their corresponding prices. Thereafter, *Reservation* component selects a ticket t_i from the list L and requests the *Payment* component to perform the payment.
All the search parameters $from$, to, $dates$, as well as the flights list L are set to low since users are not identified while sending these queries. Other sensitive data like the selected flight t_i, the price variable p and the payment parameters (identity id, credit card variable cna and code number cno) are set to high.

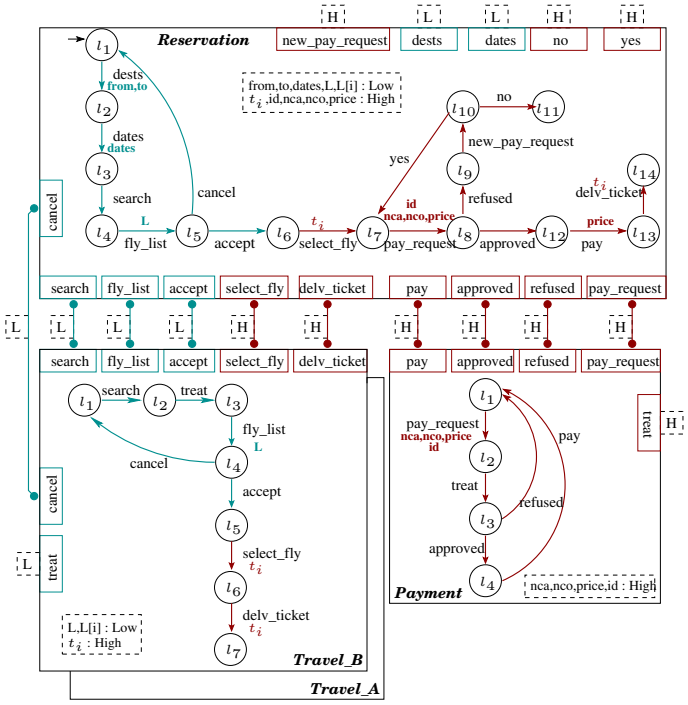

Figure 10: High-level application model.

Internal ports *dests* and *dates* as well as *search*, *fly_list*, *accept* interactions are set to low since these interactions (events) do not reveal any information about the client private trip. However, the $select\_fly$ interaction must be set to high since the observation of the selection event from *AgencyB* allows deducing the client destination. In the case of a selected flight from *AgencyA*, the $select\_fly$ interaction could be set to low since, in this case, the destination could not be deduced just from the event occurrence.

We recall that any system can be proven non-interferent iff it satisfies the syntactic security conditions from definition 6. Indeed, these conditions hold for the system model depicted in Figure 10. In particular, it can be easily checked that all assignments occurring in transitions within atomic component as well as within interactions are sequential consistent. For example, at the $select\_fly$ interaction we assign a low level security item from the flight list L to a high security level variable ti, formally $t_i = L[i]$. Besides, the security levels assignments to ports exclude inconsistencies due to causal and conflicting transitions, in all atomic components.

The decentralization of this system can be done flowing two manners of interaction partitioning: (1) conflict-free partitioning with centralized interaction management for each security level or (2) decentralized management with specific partitioning with conflict resolution between *IP* components. That is, in the first case conflicts are handled at the *IP layer* and no further interactions with *CRP layer* is needed. A solution in the second case, decentralized interaction management at each security level with arbitrary partitioning is represented by Figure 11. In this case, high security interactions are partitioned amongst IP_{1H} and IP_{2H} components while low interactions are handled by IP_L. With this partitioning, conflicts occur between ports labelling outgoing transitions at both states l_3 and l_8 of *Payment* and *Reservation* components, respectively. To handle this conflict, a centralized CRP_H for high security level interaction is introduced.

The solution proposed to automatically distribute secure systems handles well issues of protecting confidentiality in concurrent settings. From the S/R model, we automatically generates stand-alone processes (C++ code) for every S/R components (atomic, *IP* or *CRP*). S/R interactions are implemented using *TCP/IP* socket-based communication. The implementation can be deployed and run on a distributed network. For the time being, however, we assume that communications are managed through secure private/public net-

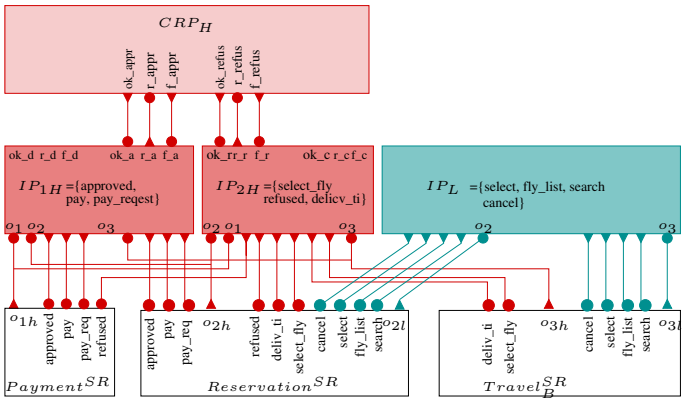

Figure 11: Distributed application model.

works, that is, communication at every security level is ensured with trusted secure channels. Nevertheless, cryptographic mechanisms that guarantee secrecy and integrity of transferred data on *TCP/IP* sockets can be included in our model and is part of our planned future work.

5 Related Work

In this work we presented a distributed *secureBIP* framework, that handles both event and data-flow non-interference, in a single semantic model. To the best of our knowledge, these properties have never been jointly considered for component-based models. Nevertheless, the need to consider together event and data flow non-interference has been recently identified in the existing literature. The bottom line is that preserving the safety of data flow in a system does not necessarily preserve safe observability on system's public behavior (i.e., secret/private executions may have an observable impact on system public events). The issue has been recently considered in [1], for data leaks and information leaks in business processes based on system's data-flows and work-flows. Also, [3] showed that formal verification of the system's event behavior is not sufficient to guarantee specific data properties. Furthermore, [10] attempted to fill the gap between respectively language-based and process calculus-based information security and make an explicit distinction between preventing the data leakage through the execution of programs and preventing secret events from being revealed in inter-process communications.

Other previous works [22, 23, 11] have studied secure program distribution. In [22, 23] the authors present a secure compiler JiF/Split to build secure distributed systems by splitting typed sequential java programs with assumption that the communications are secure through the network. Whereas [11] presents a similar approach as Jif/Split, it further enforces the network communication security by providing cryptographic mechanisms. Compared to this, we give a different approach by follow a component-based design analysis handling heterogeneous systems with formally proved transformation steps. Our method significantly reduces the verification of complex systems and automatically ensures security from the high-level model till code generation.

We should notice that recent works on information flow security in web services rely on Petri-nets modeling generated from *BPEL* orchestration tools [2]. The Petri-net graph is modified by the developer to take into account shared resources (mainly between users sessions) and security annotations are added at interaction level. Our abstract high-level model provides more fine-grained description of the system through intra-component description and a modular compositional verification. Furthermore and unlike [14], that provides limited method using calculus types refinement for web services dynamic composition and which required expensive calculations resources for types propagation for big scaled systems, our model is more general and we can use it for verifying any kind of component-based systems and not only web services.

6 Conclusion

We present an automated secure distributed implementation from abstract component-based model with multiparty interactions. Our method consists of enforcing information flow security at centralized input model then generating a distributed model where multiparty interactions are replaced with asynchronous message passing and sets of interactions belonging to same security level are handled separately. The intermediate distributed model is formally proved "secure by construction" and the generated code implements the desired security policy defined at the centralized level.
Besides, when evaluating secure systems three primary factors are usually emphasized: security, performance and ease-of-use. In our approach, security is handled at an abstract high-level and preserved till code generation automatically. Second, compared to the [8] transformation, the distribution preserves the structure of distributed S/R model where no additional components or security levels are added. Furthermore, performance is almost preserved, the only penalties being related to encryption mechanisms for network connections.
This work is now being extended in two directions. First, we are working towards the implementation of cryptographic primitives for multi-level security (MLS) systems. This implementation connects the generated C++ code to cryptographic libraries in order to ensure confidentiality and integrity of exchanged messages. Second, the current definition of transitive non-interference is relatively strict for practical use, therefore, we are investigating to which extent our decentralization method applies to relaxed version of non-interference (e.g, intransitive or with declassification mechanisms).

References

[1] R. Accorsi and A. Lehmann. Automatic information flow analysis of business process models. In *Proceedings of the 10th international conference on Business Process Management*, BPM'12, pages 172–187. Springer-Verlag, 2012. 2.3, 5

[2] R. Accorsi and C. Wonnemann. Static information flow analysis of workflow models. In *Conference on Business Process and Service Computing, volume 147 of Lecture Notes in Informatics*, pages 194–205, 2010. 5

[3] C. Bartolini, A. Bertolino, E. Marchetti, and I. Parissis. *Architecting Dependable Systems V*, chapter Data Flow-Based Validation of Web Services Compositions: Perspectives and Examples, pages 298–325. Springer, 2008. 5

[4] A. Basu, S. Bensalem, M. Bozga, J. Combaz, M. Jaber, T.-H. Nguyen, and J. Sifakis. Rigorous component-based design using the BIP framework. *IEEE Software, Special Edition – Software Components beyond Programming – from Routines to Services*, 28(3):41–48, 2011. 2, 2.1

[5] A. Basu, M. Bozga, and J. Sifakis. Modeling Heterogeneous Real-time Systems in BIP. In *Proceedings of SEFM'06*, pages 3–12. IEEE Computer Society Press, 2006. 2, 2.1

[6] E. D. Bell and J. L. La Padula. Secure computer system: Unified exposition and multics interpretation, 1976. 2.2

[7] N. Ben Said, T. Abdellatif, S. Bensalem, and M. Bozga. 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. 1, 2, 2.3, 1, 4

[8] B. Bonakdarpour, M. Bozga, M. Jaber, J. Quilbeuf, and J. Sifakis. A framework for automated distributed implementation of component-based models. *Distributed Computing*, 2012. 3, 3, 3.1, 6

[9] D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Commun. ACM*, pages 504–513, 1977. 2.2

[10] R. Focardi, S. Rossi, and A. Sabelfeld. Bridging language-based and process calculi security. In *In Proc. of Foundations of Software Science and Computation Structures (FOSSACS'05), volume 3441 of LNCS*, pages 299–315. Springer-Verlag, 2005. 2.3, 5

[11] C. Fournet, G. Le Guernic, and T. Rezk. A security-preserving compiler for distributed programs: From information-flow policies to cryptographic mechanisms. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 432–441. ACM, 2009. 5

[12] S. Frau, R. Gorrieri, and C. Ferigato. Formal aspects in security and trust. pages 210–225. Berlin, 2009. 2.3

[13] J. Goguen and J. A. Meseguer. Security policies and security models. In *Proceedings of the 1982 ieee symposium on security and privacy*, pages 11–20. IEEE Computer Society. 1, 2.2

[14] D. Hutter and M. Volkamer. Information flow control to secure dynamic web service composition. In *In International Conference on Security in Pervasive Computing*, pages 196–210, 2006. 5

[15] R. John. Noninterference, transitivity, and channel-control security policies. Technical report, dec 1992. 2.2

[16] C. K. Mani and J. Misra. *Parallel Program Design: A Foundation*. Addison Wesley Publishing Company, Inc., Reading, Massachusetts, 1988. 3.3

[17] H. Mantel. Possibilistic Definitions of Security - An Assembly Kit. In *13th IEEE Workshop on Computer Security Foundations (CSFW'00)*, page 185. IEEE Computer Society, 2000. 1

[18] D. McCullough. Noninterference and the composability of security properties. In *Security and Privacy (SP'88)*, pages 177–186. IEEE Computer Society, 1988. 1

[19] J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Security and Privacy (SP'94)*, page 79. IEEE Computer Society, 1994. 1

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

[21] A. Zakinthinos and E. S. Lee. A general theory of security properties. In *Security and Privacy (SP'97)*, pages 94–102. IEEE Computer Society, 1997. 1

[22] S. Zdancewic, L. Zheng, N. Nystrom, and A. C. Myers. Secure program partitioning. *ACM Trans. Comput. Syst.*, pages 283–328, 2002. 5

[23] L. Zheng, S. Chong, A. C. Myers, and S. Zdancewic. Using replication and partitioning to build secure distributed systems. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, SP '03, pages 236–, Washington, DC, USA, 2003. IEEE Computer Society. 5