# Invariant Preservation by Component Composition Using Semantical Interface Automata

Sebti Mouelhi, Samir Chouali, Hassan Mountassir
*Computer Science Laboratory (LIFC),*
*University of Franche-Comté, Besançon, FRANCE*
*Email: {sebti.mouelhi, samir.chouali, hassan.mountassir}@lifc.univ-fcomte.fr*

***Abstract*—Component assembly is based on the verification of the compatibility between the component interface specifications. In general, these specifications do not combine the three levels of the compatibility check: behavioral protocols, signatures, and semantics of operations. In this paper, we enrich the formalism of interface automata, used to specify component protocols, by the signatures and semantics of operations. We propose a new formalism, called "semantical interface automata" (SIAs), endowed with a stronger compositional semantics than interface automata. The semantics of operations is specified by pre and post-conditions stated over their parameters and a set of variables reflecting the behavioral conduct of components interoperability. First, we show how the component compatibility is checked at the signature, semantic, and protocol levels. Second, we establish a formal methodology to check the preservation of invariants by composition of SIAs.**

***Keywords-software components*; *interface automata*; *action semantics*; *formal correctness*; *invariants*.**

## I. INTRODUCTION

An individual component is a software unit of a third-party composition and deployment that encapsulates a set of implemented offered services and asks for a set of required ones [1]. The component interfaces depict its access points and it must be associated to a contractual specification that specify the necessary sufficient of its functional behavior at the levels of the signatures and the semantics of operations and the behavioral protocol, etc. [2], [3].

In this paper, we focus on assembling components whose behaviors are described by interface automata [4] enriched by the semantics of actions. The new formalism combines the protocol and the semantic levels of interface specifications, hence the name *semantical interface automata*. The actions of a semantical interface automaton are annotated by pre and post-conditions stated over the parameters of their correspondent operations and a set of *interface variables* shared by the automaton and its environment. The compatibility check of SIAs takes into account the action constraints specified by these conditions. Furthermore, we found a formal methodology to check correctness properties thanks to the rich interface description of SIAs. Correctness properties are typically *invariants* written in terms of the interface variables. The invariance properties are assessed at all the states of a labeled transition system representation of a SIA. In particular, we study the invariant preservation by component composition.

The paper is organized as follows. In Section II, we present the SIAs formalism features and how we check their composability and compatibility. In Section III, we explain how LTS representations are extracted from SIAs and how we check the preservation of invariants by composition of SIAs. Related works are presented in Section IV. The conclusion and future works are presented in Section V.

## II. SEMANTICAL INTERFACE AUTOMATA

Interface automata (IAs) [4], [5] have been introduced to model both the output behavior and the environment assumptions of software components. These models are non-input-enabled I/O automata [6], which means that at every state some input actions may be non-enabled. Every component interface is described by one interface automaton where input actions are used to model methods that can be called, the end of receiving messages, and the return values from such calls, as well as exception treatment. Output actions are used to model method calls, message transmissions, exceptions, and sending return values. Hidden actions represents local operations. The alphabet of an interface automaton is built of its action names annotated by "?" for input actions, by "!" for output actions, and by ";" for hidden actions.

Before defining semantical interface automata, we start by giving some preliminaries. The signature of an action $a$ is the signature of its correspondent operation implemented or solicited by the component that provokes the action. It has the form $a(i_1,...,i_n) \rightarrow (o)$ where $n \geq 0$. The set $P_a^{\mathrm{i}} = \{i_1, ..., i_n\}$ represents the set of input parameters of $a$. The set $P_a^{\mathrm{o}}$ is defined by the singleton $\{o\}$. The absence of input or output parameters is denoted by $()$. We suppose that an action has no signature if it corresponds to a return value.

Given a set of components $\mathcal{C}$, a SIA $A_c$ of a component $c$ in $\mathcal{C}$ is defined in relation to the other components in $\mathcal{C}\backslash\{c\}$. This relation is based on a set of variables $V_{\mathcal{C}}$ shared between them. We denote, by $D_w$, the domain of a variable or a parameter $w$. Given a set of variables $V$,

 ISBN: 978-1-61208-165-6 

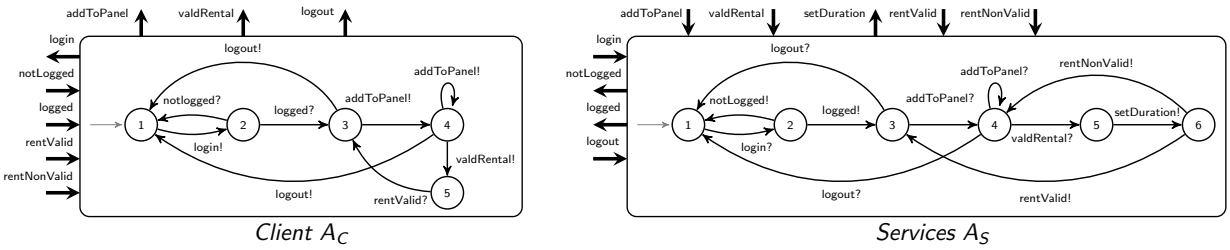


Figure 1. The semantical interface automata of the components *Client* and *Services*

*Preds*$(V)$ represents the set of first order predicates whereof free variables belong to *V*. We denote by $p'$ the predicate obtained by replacing $v$ by $v'$ in $p \in$ *Preds*$(V)$. The set *Preds*$'(V)$ is equal to $\{p' \mid p \in \textit{Preds}(V)\}$.

*Definition 1: Given a component* $c \in \mathcal{C}$, *a* semantical interface automaton $A_c = \langle S_{A_c}, i_{A_c}, \Sigma^{\mathrm{I}}_{A_c}, \Sigma^{\mathrm{O}}_{A_c}, \Sigma^{\mathrm{H}}_{A_c}, \delta_{A_c}, L_{A_c}, V_{A_c}, \mathit{Init}_{A_c}, \Psi_{A_c}\rangle$ *of c consists of*

- *a finite set* $S_{A_c}$ *of states containing an initial state* $i_{A_c}$. *A is called "empty" if* $S_A = \emptyset$;
- *three disjoint sets* $\Sigma^{\mathrm{I}}_{A_c}, \Sigma^{\mathrm{O}}_{A_c}$ *and* $\Sigma^{\mathrm{H}}_{A_c}$ *of inputs, output, and hidden actions;*
- *a set* $\delta_{A_c} \subseteq S_{A_c} \times \Sigma_{A_c} \times S_{A_c}$ *of transitions;*
- *a set* $L_{A_c}$ *of local variables and a set* $V_{A_c} \subseteq V_{\mathcal{C}}$ *of shared variables. The set* $LV_{A_c} = V_{A_c} \cup L_{A_c}$ *represents the set of all variables of* $A_c$;
- $\Psi_{A_c}$ *is a function that associates for each action* $a \in \Sigma_{A_c}$ *a tuple* $\langle \mathit{Pre}_{\Psi_{A_c}(a)}, \mathit{Post}_{\Psi_{A_c}(a)}\rangle$ *such that* $\mathit{Pre}_{\Psi_{A_c}(a)} \in \mathit{Preds}(V_{A_c} \cup P^{\mathrm{i}}_a)$ *and* $\mathit{Post}_{\Psi_{A_c}(a)} \in \mathit{Preds}(V_{A_c} \cup P^{\mathrm{i}}_a \cup P^{\mathrm{o}}_a)$;

According to Definition 1, the pre and post-conditions are defined only in terms of parameters and shared variables. They are used to verify the compatibility of two semantical interface automata, then they should be defined only on what is shared between them. The presence of local variables can be problematic because the local variables of one automaton are unknown to the others.

*Example 1:* As an example, we will consider a simple distributed multi-tier application that allows object leasing between users. The component *Services* is a server side component that plays the role of the mediator between the *Client* and the database. It provides the operations *login* that returns a reference to a persistent component *User* (the output action *logged*!) that represents the authenticated user or provokes an exception (*notLogged*!). It provides also, for the registered users, the operations *addToPanel*, *valdRental*, and *logout*. The first method makes an object to lend in the member panel, the second one validates its request to rent the objects saved in his panel if the operation *addToPanel* is called at least once, and the third one allows the member logout. The component *Services* requires the operation *setDuration* that affects a default rental duration for the chosen resource and validates totally the rental (*rentValid*!). An exception *rentNonValid* is detected if the rental validation cannot be made. In Figure 1, we show the SIAs $A_C$ and $A_S$ of the two components *Client* and *Services*.

The signatures of *Services*'s provided operations are *login*(*id*,*pass*)→(*user*), *logout*()→(), *addToPanel*(*res*)→(), *validateRental*(*ren*)→(), and *setDuration*(*beg*,*end*)→(). The parameters *id*, *beg*, and *end* are integers. The parameter *pass* is a string. The parameters *user*, *res*, and *ren*, which are references to the persistent components, are records.

According to $A_S$, *Client* can make at most two connection attempts. Elsewhere, the client connections fail. The set $\mathcal{C}$ of components is defined by {*Client*,*Services*}. The set of shared variables $V_{\mathcal{C}}$ is defined by {*sess*,*panel*}. The variable *sess* indicates the status of the client session, and the variable *panel* indicates the panel status. We assume that $V_{A_C}$ = $V_{A_S}$ = $V_{\mathcal{C}}$, $L_{A_C} = \emptyset$, and $L_{A_S} = \{satt\}$. The local variable *satt* represents the number of connection attempts accorded to clients by the component *Services*. We assume that $D_{satt}$ = $\mathbb{N}$, $D_{sess}$ = {*active*,*inactive*}, and $D_{panel}$ = {*empty*,*nonempty*}.

The semantics $\Psi_{A_C}(login)$ and $\Psi_{A_S}(login)$ of the action *login* are given in Table I as an example of the action semantics. ■

### A. Composability

Before defining the composability conditions of two SIAs, we introduce the notion of action effects, which are essential to define the concepts of full and partial control of variables by a component. Mainly, effects are used later in Section III to define the LTS representation of SIAs, but we need to introduce them at this stage to define the criteria by which the composability of two SIAs can be decided.

An effect of an action intervenes its pre and post-conditions and changes the values of shared and local variables because they are needed to check the correctness properties. We define by $\mathit{Chg}_{A_c} : \Sigma_{A_c} \rightarrow 2^{LV_{A_c}}$ the function that associates for each $a \in \Sigma_{A_c}$, the set of variables $\mathit{Chg}_{A_c}(a) \subseteq LV_{A_c}$ modifiable by $a$.

*Definition 2: An* effect *of an action a is defined by*

$$e_{A_c}(a) = \bigvee_{k \geq 1} (grd_k \wedge cmd_k \wedge Unchg_k)$$

Table I
THE SEMANTICS OF THE SHARED ACTION *login*

| *Client* $A_C$ | *Services* $A_S$ |
|---|---|
| $Pre_{\Psi_{A_C}(login)} \equiv id > 0 \wedge 8 \leq pass.length() \leq 10 \wedge sess = inactive$ | $Pre_{\Psi_{A_S}(login)} \equiv id \geq 1 \wedge 6 \leq pass.length() \leq 10 \wedge sess = inactive$ |
| $Post_{\Psi_{A_C}(login)} \equiv user.getId() = id$ | $Post_{\Psi_{A_S}(login)} \equiv user.getId() = id$ |

Table II
THE EFFECTS OF ACTIONS IN $A_C$ AND $A_S$

| **Action** | $e_{A_C}$ | $e_{A_S}$ |
|---|---|---|
| *login* | $sess = inactive \wedge Unchanged(LV_{A_C})$ | $sess = inactive \wedge ((satt \geq 0 \wedge satt' = satt - 1 \wedge sess' = sess \wedge panel' = panel) \vee Unchanged(LV_{A_S}))$ |
| *notLogged* | $sess = inactive \wedge Unchanged(LV_{A_C})$ | $sess = inactive \wedge Unchanged(LV_{A_S})$ |
| *logged* | $sess = inactive \wedge sess' = active \wedge Unchanged(\{panel\})$ | $sess = inactive \wedge 0 \leq satt < 2 \wedge sess' = active \wedge Unchanged(\{satt, panel\})$ |
| *addToPanel* | $sess = active \wedge ((panel = vide \wedge panel' = nonvide \wedge sess' = sess) \vee Unchanged(LV_{A_C}))$ | $sess = active \wedge ((panel = vide \wedge panel' = nonvide \wedge satt' = satt \wedge sess' = sess) \vee Unchanged(LV_{A_S}))$ |
| *valdRental* | $sess = active \wedge Unchanged(LV_{A_C})$ | $sess = active \wedge Unchanged(LV_{A_S})$ |
| *setDuration* | not defined | $sess = active \wedge panel = nonvide \wedge panel' = vide \wedge satt' = satt \wedge sess' = sess$ |
| *rentNonValid* | not defined | $sess = active \wedge Unchanged(LV_{A_S})$ |
| *rentValid* | $sess = active \wedge Unchanged(LV_{A_C})$ | $sess = active \wedge Unchanged(LV_{A_S})$ |
| *logout* | $sess = active \wedge sess' = inactive \wedge Unchanged(LV_{A_C} \setminus \{sess\})$ | $sess = active \wedge sess' = inactive \wedge satt' = 2$ $Unchanged(LV_{A_S} \setminus \{sess, satt\})$ |

*such that*

- *the predicate* $grd_k \in Preds(LV_{A_c})$*, for* $k \geq 1$*, is a one of the* guards *of a ;*
- $cmd_k \in Preds'(V_k)$*, where* $V_k \subseteq Chg_{A_c}(a)$*, is a* command *predicate defined in terms of primed variables* $v'$ *that represents the variables* $v \in V_k$ *after the execution of a, if* $grd_k$ *is satisfied;*
- $Unchg_k = Unchanged(LV_{A_c} \setminus V_k)$ *is a predicate defined in terms of variables in* $LV_{A_c} \setminus V_k$ *that still unchanged after the execution of a. The predicate* $Unchanged(V)$*, for a set of variables V, is*

$$\bigwedge_{v \in V} v' = v.$$

The pre and post-conditions of an action $a$ are not sufficient to define the full semantics of an action because they ignore local variables. The effect $e_{A_c}(a)$ of $a$ imposes guards $grd_k$ on the local and shared variables and, for each guard, defines a modification $cmd_k$ on variables $LV_{A_c}$.

*Example 2:* Consider the previous example, the effects of actions in $A_C$ et $A_S$ are defined in Table II. The reader can easily deduce $Chg_{A_C}$ and $Chg_{A_S}$. ■

The *fully-controlled* variables by $A_c$ are the shared variables in $V_{A_c}$ whereof $A_c$ is conscious of all the environment actions that can modify them. This requirement states that a specification must include all the actions that can modify its shared fully-controlled variables. This condition is crucial in component-based assembly, it ensures that the composite of two SIAs is consistent with both of them.

The set of *partially-controlled* variables by $A_c$ are the shared variables that can be modified by the environment actions unknown to $A_c$. We denote by $\Sigma_A^{\text{ext}} = \Sigma_A^{\text{I}} \cup \Sigma_A^{\text{O}}$ the external actions of $A$.

*Definition 3: The set* $V_{A_c}^{\text{fc}}$ *of* fully-controlled *variables by* $A_c$ *is defined by* $\{v \in V_{A_c} \mid (\forall c' \in \mathcal{C} \setminus \{c\}, a \in \Sigma_{A_{c'}} \mid v \in Chg_{A_{c'}}(a) \Rightarrow a \in \Sigma_{A_c}^{\text{ext}})\}$*. The set of* partially-controlled *variables is* $V_{A_c}^{\text{pc}} = V_{A_c} \setminus V_{A_c}^{\text{fc}}$.

*Example 3:* The variable *sess* is fully-controlled by $A_C$ and $A_S$. Contrariwise, the variable *panel* is fully-controlled by $A_S$ and partially-controlled by $A_C$ because *setDuration* $\notin \Sigma_{A_C}^{\text{ext}}$ and it changes the variable (*panel* $\in Chg_{A_S}$(*setDuration*)). ■

Given two SIAs $A_{c_1}$ and $A_{c_2}$, *Shared*($A_{c_1}$,$A_{c_2}$) = $(\Sigma_{A_{c_1}}^{\text{I}} \cap \Sigma_{A_{c_2}}^{\text{O}}) \cup (\Sigma_{A_{c_2}}^{\text{I}} \cap \Sigma_{A_{c_1}}^{\text{O}})$ is the set of shared input and output actions of $A_{c_1}$ and $A_{c_2}$. The external shared actions should have the same signatures in both $A_{c_1}$ and $A_{c_2}$.

*Definition 4: Two semantical interface automata* $A_{c_1}$ *and* $A_{c_2}$ *of two components* $c_1$ *and* $c_2$ *in* $\mathcal{C}$ *are* composable *iff*

- $\Sigma_{A_{c_1}}^{\text{I}} \cap \Sigma_{A_{c_2}}^{\text{I}} = \Sigma_{A_{c_1}}^{\text{O}} \cap \Sigma_{A_{c_2}}^{\text{O}} = \Sigma_{A_{c_1}}^{\text{H}} \cap \Sigma_{A_{c_2}} = \Sigma_{A_{c_1}} \cap \Sigma_{A_{c_2}}^{\text{H}} = \emptyset$*;*
- $L_{A_{c_1}} \cap L_{A_{c_1}} = \emptyset$*;*
- $\forall a \in Shared(A_{c_1},A_{c_2}),\ \phi \in \mathcal{I}_{V_{A_{c_1}} \cap V_{A_{c_2}}} \mid e_{A_{c_1}}(a) \wedge e_{A_{c_2}}(a)$ *is satisfiable;*

- *for all* $a \in$ *Shared*$(A_{c_1},A_{c_2})$ *whereof the signature is given by* $a(i_1,...,i_n) \rightarrow (o)$ *in* $A_{c_1}$ *and by* $a(i'_1,...,i'_n) \rightarrow (o')$ *in* $A_{c_2}$ *for all* $n \in \mathbb{N}$
  - *if* $a \in \Sigma^{\mathrm{O}}_{A_{c_1}}$, *then* $D_{i_k} \subseteq D_{i'_k}$ *for* $1 \leq k \leq n$ *and* $D_o \subseteq D_{o'}$;
  - *if* $a \in \Sigma^{\mathrm{I}}_{A_{c_1}}$, *then* $D_{i_k} \supseteq D_{i'_k}$ *for* $1 \leq k \leq n$ *and* $D_o \supseteq D_{o'}$.

The composition of two semantical interface automata $A_{c_1}$ and $A_{c_2}$ may take effect if (i) their actions are disjoint except shared input, output, hidden ones, (ii) their shared input and output actions have the same effect on variables in $V_{A_{c_1}} \cap V_{A_{c_2}}$ ($e_{A_{c_1}}(a) \wedge e_{A_{c_2}}(a)$ is satisfiable), (iii) the parameter sub-typing [7] property of actions in *Shared*$(A_{c_1},A_{c_2})$ is satisfied, and (iv) their local variables are disjoint.

*Example 4:* We have *Shared*$(A_C,A_S) = \Sigma_{A_C}$. According to Definition 4 and the indications given in the previous examples, $A_C$ and $A_S$ are composable. ■

### B. Compatibility

The semantical compatibility [8] of external shared actions and the synchronized product of two composable SIAs is defined as follows.

*Definition 5: Given an action* $a \in$ *Shared*$(A_{c_1},A_{c_2})$, *if one of the following conditions is satisfied then the action a in* $A_{c_1}$ *is* semantically compatible *with the action a in* $A_{c_2}$, *denoted by* *SComp*$_a(A_{c_1}, A_{c_2}) \equiv$ *true* (*false otherwise*):

- *if* $a \in \Sigma^{\mathrm{O}}_{A_{c_1}}$, *then (1)* *Pre*$_{\Psi_{A_{c_1}}(a)} \Rightarrow$ *Pre*$_{\Psi_{A_{c_2}}(a)}$, *and (2)* *Post*$_{\Psi_{A_{c_1}}(a)} \Leftarrow$ *Post*$_{\Psi_{A_{c_2}}(a)}$,
- *if* $a \in \Sigma^{\mathrm{I}}_{A_{c_1}}$, *then (1)* *Pre*$_{\Psi_{A_{c_1}}(a)} \Leftarrow$ *Pre*$_{\Psi_{A_{c_2}}(a)}$, *and (2)* *Post*$_{\Psi_{A_{c_1}}(a)} \Rightarrow$ *Post*$_{\Psi_{A_{c_2}}(a)}$.

*We assume that the name of parameters are the same in the semantics of actions in* $A_{c_1}$ *and* $A_{c_2}$.

The definition of the synchronized product is defined as follows.

*Definition 6: Given two components* $c_1, c_2 \in \mathcal{C}$ *whose the semantical interface automata* $A_{c_1}$ *and* $A_{c_2}$ *are composable, their product* $A_{c_1} \otimes A_{c_2}$ *is defined by*

- $S_{A_{c_1} \otimes A_{c_2}} = S_{A_{c_1}} \times S_{A_{c_2}}$; $i_{A_{c_1} \otimes A_{c_2}} = (i_{A_{c_1}}, i_{A_{c_2}})$;
- $\Sigma^{\mathrm{I}}_{A_{c_1} \otimes A_{c_2}} = (\Sigma^{\mathrm{I}}_{A_{c_1}} \cup \Sigma^{\mathrm{I}}_{A_{c_2}}) \setminus$ *Shared*$(A_{c_1}, A_{c_2})$;
- $\Sigma^{\mathrm{O}}_{A_{c_1} \otimes A_{c_2}} = (\Sigma^{\mathrm{O}}_{A_{c_1}} \cup \Sigma^{\mathrm{O}}_{A_{c_2}}) \setminus$ *Shared*$(A_{c_1}, A_{c_2})$;
- $\Sigma^{\mathrm{H}}_{A_{c_1} \otimes A_{c_2}} = \Sigma^{\mathrm{H}}_{A_{c_1}} \cup \Sigma^{\mathrm{H}}_{A_{c_2}} \cup \{a \in$ *Shared*$(A_{c_1}, A_{c_2}) \mid$ *SComp*$_a(A_{c_1}, A_{c_2}) \equiv$ *true*$\}$;
- $L_{A_{c_1} \otimes A_{c_2}} = L_{A_{c_1}} \cup L_{A_{c_2}}$; $V_{A_{c_1} \otimes A_{c_2}} = V_{A_{c_1}} \cup V_{A_{c_2}}$;
- $((s_1, s_2), a, (s'_1, s'_2)) \in \delta_{A_{c_1} \otimes A_{c_2}}$ iff
  - $a \notin$ *Shared*$(A_{c_1}, A_{c_2}) \wedge (s_1, a, s'_1) \in \delta_{A_{c_1}} \wedge s_2 = s'_2$,
  - $a \notin$ *Shared*$(A_{c_1}, A_{c_2}) \wedge (s_2, a, s'_2) \in \delta_{A_{c_2}} \wedge s_1 = s'_1$,
  - $a \in$ *Shared*$(A_{c_1}, A_{c_2}) \wedge (s_1, a, s'_1) \in \delta_{A_{c_1}} \wedge (s_2, a, s'_2) \in \delta_{A_{c_2}} \wedge$ *SComp*$_a(A_{c_1}, A_{c_2}) \equiv$ *true;*
- $\Psi_{A_{c_1} \otimes A_{c_1}}$ *is defined by:*
  - $\Psi_{A_{c_i}}$ *for* $a \in \Sigma_{A_{c_i}} \setminus$ *Shared*$(A_{c_1}, A_{c_2})$ *for* $i \in \{1, 2\}$;
  - $\langle$*Pre*$_{\Psi_{A_{c_1}}(a)}$, *Post*$_{\Psi_{A_{c_2}}(a)}\rangle$ *for* $a \in$ *Shared*$(A_{c_1}, A_{c_2}) \cap \Sigma^{\mathrm{O}}_{A_{c_1}}$ *such that* *SComp*$_a(A_{c_1}, A_{c_2}) \equiv$ *true;*
  - $\langle$*Pre*$_{\Psi_{A_{c_2}}(a)}$, *Post*$_{\Psi_{A_{c_1}}(a)}\rangle$ *for* $a \in$ *Shared*$(A_{c_1}, A_{c_2}) \cap \Sigma^{\mathrm{I}}_{A_{c_1}}$ *such that* *SComp*$_a(A_{c_1}, A_{c_2}) \equiv$ *true.*

We assume that $e_{A_{c_1} \otimes A_{c_1}}$ is defined by $e_{A_{c_1}}(a) \wedge e_{A_{c_2}}(a)$ for all $a \in$ *Shared*$(A_{c_1}, A_{c_2})$, by $e_{A_{c_1}}(a)$ for all $a \in \Sigma_{A_{c_1}} \setminus$ *Shared*$(A_{c_1}, A_{c_2})$, and by $e_{A_{c_2}}(a)$ for all $a \in \Sigma_{A_{c_2}} \setminus$ *Shared*$(A_{c_1}, A_{c_2})$.

The incompatibility between $A_{c_1}$ and $A_{c_2}$ is due to (i) the existence of some *illegal states* $(s_1,s_2)$ in the set of transitions $\delta_{A_{c_1} \otimes A_{c_1}}$ where one of the two SIAs $A_{c_1}$ and $A_{c_2}$ outputs a shared action *a* from $s_1$, which is not accepted as input from $s_2$ or vice versa, or (ii) from that states they synchronize on the action *a* but *SComp*$_a(A_{c_1}, A_{c_2}) \equiv$ *false*.

*Definition 7: The set of* illegal states *Illegal*$(A_{c_1}, A_{c_2}) \subseteq S_{A_{c_1}} \times S_{A_{c_2}}$ *is defined by* $\{(s_1, s_2) \in S_{A_{c_1} \otimes A_{c_2}} \mid (\exists\ a \in$ *Shared*$(A_{c_1}, A_{c_2}) \mid (C_1 \vee C_2$ *holds*$))\}$

$$C_1 = \begin{pmatrix} (a \in \Sigma^{\mathrm{O}}_{A_{c_1}}(s_1) \wedge a \notin \Sigma^{\mathrm{I}}_{A_{c_2}}(s_2)) \vee (a \in \Sigma^{\mathrm{O}}_{A_{c_1}}(s_1) \wedge \\ a \in \Sigma^{\mathrm{I}}_{A_{c_2}}(s_2) \wedge \textit{SComp}_a(A_{c_1}, A_{c_2}) \equiv \textit{false}) \end{pmatrix}$$

$$C_2 = \begin{pmatrix} (a \in \Sigma^{\mathrm{O}}_{A_{c_2}}(s_2) \wedge a \notin \Sigma^{\mathrm{I}}_{A_{c_1}}(s_1)) \vee (a \in \Sigma^{\mathrm{O}}_{A_{c_2}}(s_2) \wedge \\ a \in \Sigma^{\mathrm{I}}_{A_{c_1}}(s_1) \wedge \textit{SComp}_a(A_{c_1}, A_{c_2}) \equiv \textit{false}) \end{pmatrix}$$

The reachability of states in *Illegal*$(A_{c_1}, A_{c_2})$ do not implies that $A_{c_1}$ and $A_{c_2}$ are not compatible. The existence of an environment *E* (a semantical interface automaton) that produces appropriate inputs for $A_{c_1} \otimes A_{c_2}$ ensures that illegal states are not reached. The compatible states, denoted by *Comp*$(A_{c_1}, A_{c_2})$, are states in which $A_{c_1} \otimes A_{c_2}$ avoids reaching illegal states by enabling output or internal actions (optimistic approach) [4].

*Definition 8:* $A_{c_1}$ *and* $A_{c_2}$ *are* compatible *iff the initial state of* $A_{c_1} \otimes A_{c_2}$ *is compatible.*

The verification steps [4] of the compatibility between $A_{c_1}$ and $A_{c_2}$ without considering the semantics of actions are listed below.

**Algorithm**
**Input** : Two SIAs $A_{c_1}$ and $A_{c_2}$.
**Output** : $A_{c_1} \parallel A_{c_2}$.
**Algorithm steps :**

1) compute the product $A_{c_1} \otimes A_{c_2}$,
2) compute *Illegal*$(A_{c_1}, A_{c_2})$,
3) compute the set of incompatible states in $A_{c_1} \otimes A_{c_2}$: the states from which the illegal states are reachable by enabling only internal and output actions,
4) compute the composition $A_{c_1} \parallel A_{c_2}$ by eliminating from the automaton $A_{c_1} \otimes A_{c_2}$, the illegal state, the

 ISBN: 978-1-61208-165-6 

incompatible states, and the unreachable states from the initial state,

5) if $A_{c_1} \parallel A_{c_2}$ is empty then $A_{c_1}$ and $A_{c_2}$ are not compatible, therefore $c_1$ and $c_2$ can not be assembled correctly in any environment. Otherwise, $A_{c_1}$ and $A_{c_2}$ are compatible.

Our approach increases the complexity[1] of the previous proposed one by taking into account the semantic compatibility check of actions in *Shared*$(A_1, A_2)$, whereof the complexity is determined by the logic and the context theories within the formulas are defined.

*Example 5:* According to Table I and II and the previous definitions and examples, $A_C$ and $A_S$ are compatible if *SComp*$_a$($A_C$,$A_S$) $\equiv$ *true* for all $a \in$ *Shared*($A_C$,$A_S$). ■

*Theorem 1: The composition $\parallel$ between SIAs is a commutative and associative operation.*

*Proof:* The proof is based on that presented in [5] by considering the action semantics. ■

## III. Checking Invariance Properties

In this section, we found formal methodology to design and check the correctness properties of semantical interface automata thanks to their rich semantics based on the use of variables. The correctness properties, as said in the beginning of the paper, are invariants. Commonly, specifiers have to model a system by a transition system to check invariants and other types of system temporal properties. The states of a such transition system are associated to a set of atomic propositions stated on a set of modifiable variables. Our contribution follows a similar procedure. A SIA $A$ is translated to a labeled transition system $\mathit{LTS}(A)$ whose states are the variable valuations. Starting from an initial valuation, the effects of actions update the variables and $\mathit{LTS}(A)$ is generated.

### A. LTS representation

Before defining the LTS representations of a SIA, we start by defining some preliminaries. A *valuation* of a set of variables *V* is defined by

$$\phi : V \rightarrow \bigcup_{v_i \in V} D_{v_i}$$

that associates to each $v_i \in V$ a value in $D_{v_i}$. We denote by $\phi\langle V' \rangle$ the restriction of $\phi$ to the set $V' \subseteq V$. The set $\mathcal{I}_V$ is the set of all possible valuations $\phi$ in *V*. Given an action $a \in \Sigma_A$, we denote by $\mathcal{E}(\phi, e_A(a)) \in \mathcal{I}_{LV_A}$ the valuation of variables $LV_A$ after the execution of *a* for a valuation $\phi$ of $LV_A$.

The LTS representation of a SIA $A$ transforms its set of transitions to a labeled transition system whereof the states belong to $\mathcal{I}_{LV_A}$ and the labels are the actions in $\Sigma_A$. The LTS representations of SIAs allows the separation between the task of checking interoperability and that of the correctness properties check. It's clear that a SIA has fewer states than its LTS representation, which allow to reduce the complexity of the interoperability checking. The LTS representations are devoted to check correctness properties.

*Definition 9: The* LTS representation $\mathit{LTS}(A) = \langle S_{\mathit{LTS}(A)},\ I_{\mathit{LTS}(A)},\ \Sigma_{\mathit{LTS}(A)},\ \delta_{\mathit{LTS}(A)} \rangle$ *of a semantical interface automata A is a labeled transition system defined by*

- $S_{\mathit{LTS}(A)} \subseteq \mathcal{I}_{LV_A}$;
- $I_{\mathit{LTS}(A)} = \mathit{Init}_A$ *where* $\mathit{Init}_A$ *is the initial valuation of* $LV_A$;
- $\Sigma_{\mathit{LTS}(A)} = \Sigma_A$;
- $(\phi_1, a, \phi_2) \in \delta_{\mathit{LTS}(A)}$ *iff* $\phi_1 \in \mathcal{I}_{LV_A}$, *for* $s \in S_A$, $a \in \Sigma_A(s)$, *and* $\phi_2 = \mathcal{E}(\phi_1, e_A(a))$.

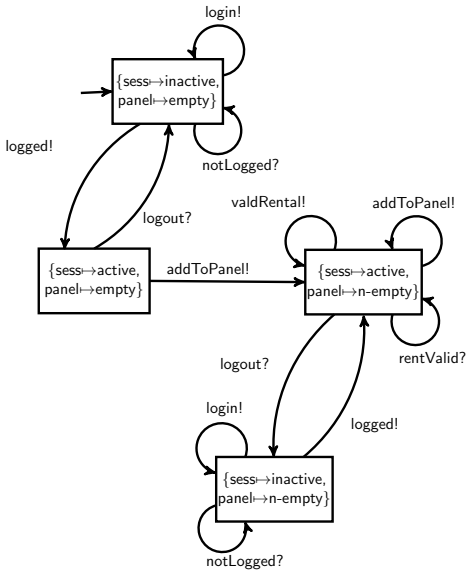


Figure 2. The LTS representation $\mathit{LTS}(A_C)$ of $A_C$

*Example 6:* The LTS representation $\mathit{LTS}(A_C)$ of the semantical interface automaton $A_C$ of the component *Client* is shown in Figure 2 where $\mathit{Init}_{A_C}$ = {*sess* $\mapsto$ *inactive*, *panel* $\mapsto$ *empty*}. ■

Given two SIAs $A_1$ and $A_2$, the following property establishes that for each state $\phi \in S_{\mathit{LTS}(A_1 \| A_2)}$, only the valuations of fully-controlled variables of both $A_1$ and $A_2$ are the same in $A_1$, $A_2$ et $A_1 \parallel A_2$. These variables are modifiable exclusively by actions in *Shared*$(A_1, A_2)$. We denote by $LV_A^{\mathrm{fc}}$, the set $V_A^{\mathrm{fc}} \cup L_A$ of a SIA *A*.

*Property 1: Given* $W = LV_{A_1 \| A_2}$, $V = LV_{A_1}^{\mathrm{fc}} \cap LV_{A_2}^{\mathrm{fc}}$, $V' = LV_{A_1}^{\mathrm{fc}} \cup LV_{A_2}^{\mathrm{fc}}$, $V_1 = LV_{A_1}^{\mathrm{fc}} \setminus LV_{A_2}^{\mathrm{fc}}$, *and* $V_2 = LV_{A_2}^{\mathrm{fc}} \setminus LV_{A_1}^{\mathrm{fc}}$ *where* $A_1$ *and* $A_2$ *are two compatible SIAs*

[1]The complexity of checking the compatibility between two interface automata $A_1$ and $A_2$ is in time linear on $|A_1|$ and $|A_2|$ [4].

*and* $\mathit{Init}_{A_1}\langle V_{A_1} \cap V_{A_2}\rangle = \mathit{Init}_{A_2}\langle V_{A_1} \cap V_{A_2}\rangle$*, for all* $\phi \in S_{\mathit{LTS}(A_1 \| A_2)}$*, there exists* $\phi_1 \in S_{\mathit{LTS}(A_1)}$ *and* $\phi_2 \in S_{\mathit{LTS}(A_2)}$ *such that*

1) $\phi\langle V\rangle = \phi_1\langle V\rangle = \phi_2\langle V\rangle$ *knowing that* ($V = LV^{\mathrm{fc}}_{A_1} \cap LV^{\mathrm{fc}}_{A_2} = V^{\mathrm{fc}}_{A_1} \cap V^{\mathrm{fc}}_{A_2}$)*;*
2) $\phi\langle V'\rangle$ *is defined as follows :*

$$\begin{cases} \phi\langle V\rangle & \text{for all } v \in V; \\ \phi_1\langle V_1\rangle & \text{for all } v \in V_1; \\ \phi_2\langle V_2\rangle & \text{for all } v \in V_2. \end{cases}$$

### B. Invariant specification

Given an LTS representation $\mathit{LTS}(A)$ of a SIA $A$, we denote by $\varphi[V] \in \mathit{Preds}(V)$, a first order formula whose free variables belong to $V \subseteq LV_A$. A formula $\varphi[V]$ is satisfied at the state $\phi \in S_{\mathit{LTS}(A)}$ iff $\phi$ satisfies $\varphi[V]$, i.e., the following condition is satisfied:

$$(\bigwedge_{v \in LV_A} v = \phi(v)) \Rightarrow \varphi[V].$$

*Definition 10: Given an LTS representation* $\mathit{LTS}(A)$ *of a SIA A and a set* $V \subseteq LV_A$*, an* invariant $\varphi[V]$ *of* $\mathit{LTS}(A)$ (*written* $\mathit{LTS}(A) \models \varphi[V]$) *is a first order formula such that for all* $\phi \in S_{\mathit{LTS}(A)}$*,* $\phi\langle V\rangle$ *satisfies* $\varphi[V]$ ($\phi\langle V\rangle \models \varphi[V]$) *and* $\phi$ *satisfies* $\varphi[V]$ ($\phi \models \varphi[V]$).

*Example 7:* The predicate $\varphi[LV_{A_S}]$ = ¬(*satt* = 2 ∧ *sess* = *active*) is an invariant of $\mathit{LTS}(A_S)$ shown in Figure 2. ■

### C. Invariant preservation by composition

The following theorem establishes that only invariants stated on local and fully-controlled variables can be preserved by the LTS representations of the composition $A_1 \parallel A_2$ of two compatible SIAs $A_1$ and $A_2$ because they are aware of all the environment actions that can modify these variables. An invariant stated on partially-controlled variables of $A_1$ (resp. $A_2$) cannot be preserved by composition because it is possible that $A_2$ (resp. $A_1$) modifies the values by actions unknown to $A_1$ (resp. $A_2$) in such way the invariant is violated. Corollary 2 can easily be deduced from that theorem.

*Theorem 2 (Invariant Preservation by SIA Composition): Given two* $\mathit{LTS}(A_1)$ *and* $\mathit{LTS}(A_2)$ *respectively of two compatibles SIAs* $A_1$ *and* $A_2$ *and* $\mathit{Init}_{A_1}\langle V_{A_1} \cap V_{A_2}\rangle = \mathit{Init}_{A_2}\langle V_{A_1} \cap V_{A_2}\rangle$*, for all* $\varphi_1[LV^{\mathrm{fc}}_{A_1}]$ *and* $\varphi_2[LV^{\mathrm{fc}}_{A_2}]$*, if* $\mathit{LTS}(A_1) \models \varphi_1[LV^{\mathrm{fc}}_{A_1}]$ *and* $\mathit{LTS}(A_2) \models \varphi_2[LV^{\mathrm{fc}}_{A_2}]$*, then* $\mathit{LTS}(A_1 \parallel A_2) \models (\varphi_1[LV^{\mathrm{fc}}_{A_1}] \wedge \varphi_2[LV^{\mathrm{fc}}_{A_2}])$.

*Proof:* We have the following assumptions:

1) $\forall \phi_1 \in S_{\mathit{LTS}(A_1)} \mid \phi_1\langle LV^{\mathrm{fc}}_{A_1}\rangle \models \varphi_1[LV^{\mathrm{fc}}_{A_1}]$ and $\phi_1 \models \varphi_1[LV^{\mathrm{fc}}_{A_1}]$;
2) $\forall \phi_2 \in S_{\mathit{LTS}(A_2)} \mid \phi_2\langle LV^{\mathrm{fc}}_{A_2}\rangle \models \varphi_2[LV^{\mathrm{fc}}_{A_2}]$ and $\phi_2 \models \varphi_2[LV^{\mathrm{fc}}_{A_2}]$;

We have to prove that, for all $\phi \in S_{\mathit{LTS}(A_1 \| A_2)}$, $\phi\langle LV^{\mathrm{fc}}_{A_1} \cup LV^{\mathrm{fc}}_{A_2}\rangle \models (\varphi_1[LV^{\mathrm{fc}}_{A_1}] \wedge \varphi_2[LV^{\mathrm{fc}}_{A_2}])$ and $\phi \models (\varphi_1[LV^{\mathrm{fc}}_{A_1}] \wedge \varphi_2[LV^{\mathrm{fc}}_{A_2}])$? According to the property 1(2), we have for all $\phi \in S_{\mathit{LTS}(A_1 \| A_2)}$, there exists $\phi_1 \in S_{\mathit{LTS}(A_1)}$ and $\phi_2 \in S_{\mathit{LTS}(A_2)}$ such that $\phi\langle LV^{\mathrm{fc}}_{A_1} \cup LV^{\mathrm{fc}}_{A_2}\rangle$ is equal to

$$\begin{cases} \phi\langle LV^{\mathrm{fc}}_{A_1} \cap LV^{\mathrm{fc}}_{A_2}\rangle & \forall v \in LV^{\mathrm{fc}}_{A_1} \cap LV^{\mathrm{fc}}_{A_2}; \\ \phi_1\langle LV^{\mathrm{fc}}_{A_1} \setminus LV^{\mathrm{fc}}_{A_2}\rangle & \forall v \in LV^{\mathrm{fc}}_{A_1} \setminus LV^{\mathrm{fc}}_{A_2}; \\ \phi_2\langle LV^{\mathrm{fc}}_{A_2} \setminus LV^{\mathrm{fc}}_{A_1}\rangle & \forall v \in LV^{\mathrm{fc}}_{A_2} \setminus LV^{\mathrm{fc}}_{A_1}. \end{cases}$$

We can deduce, according to the property 1(1), that

$$\phi\langle LV^{\mathrm{fc}}_{A_1} \cup LV^{\mathrm{fc}}_{A_2}\rangle = \begin{cases} \phi_1\langle LV^{\mathrm{fc}}_{A_1}\rangle & \text{for all } v \in LV^{\mathrm{fc}}_{A_1}; \\ \phi_2\langle LV^{\mathrm{fc}}_{A_2}\rangle & \text{for all } v \in LV^{\mathrm{fc}}_{A_2}. \end{cases}$$

We have $\phi_1\langle LV^{\mathrm{fc}}_{A_1}\rangle \models \varphi_1[LV^{\mathrm{fc}}_{A_1}]$ (assumption 1) and $\phi_2\langle LV^{\mathrm{fc}}_{A_2}\rangle \models \varphi_2[LV^{\mathrm{fc}}_{A_2}]$ (assumption 2), then we can deduce that $\phi\langle LV^{\mathrm{fc}}_{A_1} \cup LV^{\mathrm{fc}}_{A_2}\rangle \models \varphi_1[LV^{\mathrm{fc}}_{A_1}]$ and $\phi\langle LV^{\mathrm{fc}}_{A_1} \cup LV^{\mathrm{fc}}_{A_2}\rangle \models \varphi_2[LV^{\mathrm{fc}}_{A_2}]$. Consequently, $\phi\langle LV^{\mathrm{fc}}_{A_1} \cup LV^{\mathrm{fc}}_{A_2}\rangle \models (\varphi_1[LV^{\mathrm{fc}}_{A_1}] \wedge \varphi_2[LV^{\mathrm{fc}}_{A_2}])$ and $\phi \models (\varphi_1[LV^{\mathrm{fc}}_{A_1}] \wedge \varphi_2[LV^{\mathrm{fc}}_{A_2}])$. ■

*Corollary 1: Given two* $\mathit{LTS}(A_1)$ *and* $\mathit{LTS}(A_2)$ *of two compatibles SIAs* $A_1$ *and* $A_2$ *such that the assumptions of Theorem 2 are satisfied, for all* $\varphi[V^{\mathrm{fc}}_{A_1} \cap V^{\mathrm{fc}}_{A_2}]$*, if* $\mathit{LTS}(A_1) \models \varphi[V^{\mathrm{fc}}_{A_1} \cap V^{\mathrm{fc}}_{A_2}]$ *and* $\mathit{LTS}(A_2) \models \varphi[V^{\mathrm{fc}}_{A_1} \cap V^{\mathrm{fc}}_{A_2}]$*, then* $\mathit{LTS}(A_1 \parallel A_2) \models \varphi[V^{\mathrm{fc}}_{A_1} \cap V^{\mathrm{fc}}_{A_2}]$.

*Example 8:* The predicate $\varphi[LV^{\mathrm{fc}}_{A_S}]$ = ¬(*satt* = 2 ∧ *sess* = *active*) is an invariant of $\mathit{LTS}(A_S)$. We can deduce that $\mathit{LTS}(A_C \parallel A_S) \models$ ¬(*satt* = 2 ∧ *sess* = *active*). ■

## IV. Related Works

Luca de Alfaro and al. [9] has proposed "sociable" interface modules SIMs to specify component interfaces. The formalism communicates via both actions and shared variables and the synchronization between actions is based on two main principles: (i) the first principle is that the same actions can label both input and output transitions, and (ii) the second is that global variables can be updated by multiple interfaces. The authors show that the compatibility and the refinement check of SIMs can be made thanks to efficient symbolic algorithms implemented in the tool TICC [10] (Tool for Interface Compatibility).

The main differences between SIMs and our proposed approach can be identified in the following points. First, in SIMs, internal actions are not considered. They concertize the local behaviors of components and the synchronization of shared input and output actions. Internal actions are necessary to develop closed systems composed of a prefixed set of components. They can be also useful when a component instance is associated with a specific client. For example, the EJB *stateful* session beans cannot be composed with many clients. Shared input and output actions disappear (become internal) in the composition between a stateful the session bean instance and its client in such way other clients cannot be connected to the bean using the same shared actions. Thus, SIAs can be applied to specify both closed and open

component-based systems by cons, SIMs are rather relevant to specify only open systems.

Second, in SIMs, a component could require and offer the same service (an action can label both input and output transitions). In our approach, this type of components is not considered. Furthermore, in our approach, we explicitly define to what input and output actions correspond (operation calls, receiving return values, exceptions, etc) which is not the case in SIMs.

Third, we demonstrate that only invariants stated on fully-controlled variables (history variables in SIMs) can be preserved by composition. The authors in [9] do not take into account this issue. Finally, SIAs unifies the use of operation parameters and variables to describe the compositional semantics of components. In addition, we show that we can separate protocols, used commonly to verify component compatibility and composition from models used to check correctness. SIAs (simple states, actions, and semantics) are used to check the component compatibility and their LTS representations are used to check correctness properties preservation by composition.

Ivana Černá and al. [11] have founded "Component-interaction automata" (CoIN) to reason about the behavioral aspects of component-based systems by respecting a given architecture. They also proposed methods to check component assembly correctness by verifying properties, like consequences of operation calls and fairness without using variables. These properties are expressed in an extended version of the linear temporal logic called CI-LTL and verified using model-checking techniques [12].

The approach proposed in [13] is a formal methodology for describing behavioral protocols of interacting, concurrent components with data states. The authors describes component protocols by means of labeled transition systems, which specify the scheduling of operation calls and the data states updates by using of pre and post-conditions. Furthermore, they endow protocols with a model-theoretic semantics describing the class of all correct implementations (refinement) of an abstract protocol.

In [14], the authors have proposed an approach endowing Sun's Enterprise JavaBeans (EJB) component by behavioral protocols. The proposed framework provides a set of mechanisms allowing the automated extraction of protocols from EJB components and the verification of coherence between these protocols. Protocols are represented by particular labeled transition systems.

## V. Conclusion and Future Works

In this paper, we propose a formalism based on interface automata enriched by the use of the action semantics to describe behavioral protocols of components and to check their compatibility and safety. In particular, We study the problem of invariant preservation by composition. In the future, we intent to adapt the alternating simulation approach used to refine interface automata to support the treatment of the action semantics and to ensure the requirement of invariant preservation also by refinement.

## References


[1] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*. Boston, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[2] D. Konstantas, *Interoperation of object-oriented applications*. Hertfordshire, UK: Prentice Hall International Ltd., 1995, pp. 69–95.

[3] P. Wegner, "Interoperability," *ACM Comput. Surv.*, vol. 28, pp. 285–287, March 1996.

[4] L. de Alfaro and T. A. Henzinger, "Interface automata," *SIGSOFT Softw. Eng. Notes*, vol. 26, no. 5, pp. 109–120, 2001.

[5] L. d. Alfaro and T. A. Henzinger, "Interface-based design," in *Engineering Theories of Software-intensive Systems*. Springer, 2005, pp. 83–104.

[6] N. A. Lynch and M. R. Tuttle, "Hierarchical correctness proofs for distributed algorithms," in *PODC '87: Proc. of the 6th ACM Symp. on principles of distributed computing*. New York, USA: ACM, 1987, pp. 137–151.

[7] B. H. Liskov and J. M. Wing, "A behavioral notion of subtyping," *ACM Trans. Program. Lang. Syst.*, vol. 16, pp. 1811–1841, November 1994.

[8] S. Heiler, "Semantic interoperability," *ACM Comput. Surv.*, vol. 27, pp. 271–273, June 1995.

[9] L. d. Alfaro, L. D. d. Silva, M. Faella, A. Legay, P. Roy, and M. Sorea, "Sociable interfaces," in *The Proc. 5th Int. WS. on Frontiers of Combining Systems, LNAI 3717*. Springer-Verlag, 2005, pp. 81–105.

[10] L. d. Alfaro, L. D. d. Silva, M. Faella, A. Legay, V. Raman, and P. Roy, "Ticc: A tool for interface compatibility and composition," in *The Proc. 18th Int. Conf. on Computer Aided Verification (CAV), volume 4144 of LNCS*. Springer, 2006, pp. 59–62".

[11] B. Zimmerova, P. Vařeková, N. Beneš, I. Černá, L. Brim, and J. Sochor, "The common component modeling example." Berlin, Heidelberg: Springer-Verlag, 2008, ch. Component-Interaction Automata Approach (CoIn), pp. 146–176.

[12] J. Barnat, L. Brim, I. Černá, P. Moravec, P. Ročkai, and P. Šimeček, "Divine - a tool for distributed verification," in *Computer Aided Verification*, ser. LNCS. Springer Berlin / Heidelberg, 2006, vol. 4144, pp. 278–281.

[13] S. S. Bauer, R. Hennicker, and S. Janisch, "Behaviour protocols for interacting stateful components," *Electron. Notes Theor. Comput. Sci.*, vol. 263, pp. 47–66, June 2010.

[14] A. Farías and M. Südholt, "On components with explicit protocols satisfying a notion of correctness by construction," in *Confederated Int. Conf. DOA, CoopIS and ODBASE 2002*. London, UK: Springer-Verlag, 2002, pp. 995–1012.


 ISBN: 978-1-61208-165-6 