A compositional synthesis of failure-free connectors for correct components assembly

Paola Inverardi
University of L'Aquila
Dip. Informatica
via Vetoio 1, 67100 L'Aquila, Italy
inverard@di.univaq.it

Massimo Tivoli
University of L'Aquila
Dip. Informatica
via Vetoio 1, 67100 L'Aquila, Italy
tivoli@di.univaq.it

ABSTRACT

Correct automatic assembly of software components is considered an important issue of CBSE (*Component-Based Software Engineering*). It is related to the ability to establish properties on the assembly code by only assuming a relative knowledge of the single components properties. In our precedent works, we have provided our answer to this problem by discussing a software architecture based approach in which the software architecture imposed on the assembly allows for detection and recovery of COTS (*Commercial-Off-The-Shelf*) integration anomalies. One of the crucial aspects of our assembly technique is related to the ability to synthesize a specification-satisfying assembly code (i.e. the failures-free connector) in such a way that the synthesis results compositional with respect to system evolutions. That is every time the system evolves, in order to automatically synthesize the failures-free connector for the new version of the specification-satisfying system it is enough to repeat the synthesis only for the part of the system related to its evolution.

1. INTRODUCTION

Correct automatic assembly of software components is considered an important issue of CBSE [4] (*Component-Based Software Engineering*). It is related to the ability to establish properties on the assembly code by only assuming a relative knowledge of the single components properties. In our precedent works [7, 5, 11, 8, 6], we have developed an architectural approach to the assembly problem. This approach is to generate systems by assuming a well defined architectural style [8] in such a way that it is possible to detect and to fix software behavioral anomalies. We assume that a high-level description of the desired assembled system is available and that a precise definition of the properties to satisfy exists. With these assumptions we have developed a framework that automatically derives the assembly code for a set of components so that, if possible, a property-satisfying system is generated. The assembly code implements an explicit software connector which mediates all interactions among the system components acting as a new component to be inserted in the composed system. The connector can then be analyzed and modified in such a way that the behavioral (i.e. functional) properties of the composed system are satisfied. Depending on the kind of property, the analysis of the connector only is enough to obtain a property satisfying version of the system. Otherwise, the property is due to some component internal behavior and cannot be fixed without directly operating on the component code. In a component based setting in which we are assuming black-boxes components, this is the best we can expect to do.

An important aspect of our assembly technique is related to the ability to synthesize the specification-satisfying version of the composed system in such a way that the synthesis results compositional with respect to system evolutions. That is every time the system evolves, in order to automatically synthesize the failures-free connector for the new version of the specification-satisfying system it is enough to repeat the synthesis for the part of the system related to its evolution. Let us suppose that we have synthesized a component-based system by automatically assembling n components through a connector in order to satisfy the system's specification. Moreover, let us suppose that later on we add m new components in order to add functionalities to the system. Then it is desirable to perform the synthesis for the sub-system made of the new m components by retaining the behavioral properties validated for the old n components and by enforcing only the new specified behavioral properties. Thus by exploiting the old failures-free connector, we can derive the specification-satisfying assembly code (i.e. a new failure-free connector) for the whole new system made of the old n plus the new m components.

In this paper we show by using an explanatory example that our approach is compositional with respect to the assembled system evolution. In our context this means that we are compositional with respect to: i) the automatic synthesis of the new connector representing the glue code for the components constituting the new version of the system and ii) the set of behavioral properties to be validated; this set is composed by the set of the new specified behavioral properties and the set of the old behavioral properties already validated for the old version of the system (i.e the system

based on the old connector).

We define a connector composition technique that synthesizes a new connector as a function of an already existent connector. The new connector is automatically synthesized in such a way that the behavioral properties satisfied by the already existent connector are maintained. Thus the new specified behavioral properties are the only properties to be enforced on the new connector. This makes our approach compositional. The problem we want to treat can be phrased as follows: *Let S be a system composed of $C_1, .., C_n$ components plus a connector K; let K be the assembly code for the n components such that it satisfies a set of behavioral properties P; build a new connector* $K' = F(K, C_1, .., C_n, C_{n+1}, .., C_{n+m})$ *in order to add the components $C_{n+1}, .., C_{n+m}$ to S in such a way that K' satisfies each property in P and each one in a new specified set P' of behavioral properties.* The function F takes a connector K representing the properties-satisfying assembly code for $C_1, .., C_n$ and returns a connector K' representing the properties-satisfying assembly code for $C_1, .., C_n, C_{n+1}, .., C_{n+m}$. F is formally defined in Section 5.

The paper is organized as follows. Section 2 provides background notions to understand our approach to the assembly problem [7, 5, 11, 8, 6]. Section 3 describes a family of component-based systems associated to the problem we want to treat. Section 4 briefly reports our approach in order to show in Section 5 its compositional nature. Section 6 discusses future works and concludes.

2. BACKGROUND

We consider component-based software systems that reflect an architectural style called *Connector Based Architecture* (CBA) [8]. This style consists of components and connectors. Referring to [12], we consider a software component as *"a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."* We operate in the domain of *black-box reuse* which refers to the concept of reusing implementations without relying on anything but their interfaces and specifications. Referring to [3], we consider a software connector as a unit of coordination that mediates the communication, cooperation and interaction among components. Moreover, in CBA style components and connectors define a notion of *top* and *bottom*. The top (bottom) of a component may be connected to the bottom (top) of a single connector. Components can only communicate via connectors. It is disallowed the direct connection between connectors. Components communicate synchronously by passing two type of messages: notifications and requests. A notification is sent downward, while a request is sent up. Connectors are responsible for the routing of messages and they exhibit a strictly sequential input-output behavior[1]. CBA style is a generic layered style, for the sake of presentation we will only deal with single layered systems. In [8] we show how to cope with multi-layered systems. In Figure 1 we show an instance of CBA style made of two components and one connector.

[1]Each input action is strictly followed by the corresponding output action.

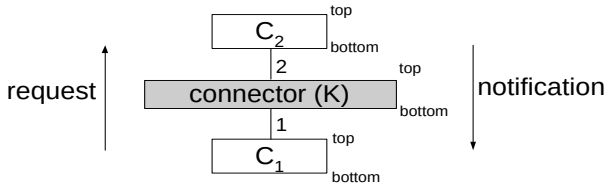

Figure 1: An instance of CBA style

3. CONFIGURATION FORMALIZATION

To our purposes we need to formalize two different ways to compose a system. The first one is called *Connector Free Architecture* (CFA) and is defined as *a set of components directly connected in a synchronous way* (i.e. without a connector). The second one is called *Connector Based Architecture* (CBA) and is defined as *a set of components directly connected in a synchronous way to one or more connectors.* In order to describe components and system behaviors we use CCS [9] (*Calculus of Communicating Systems*) notation. Our framework allows to automatically derive these CCS descriptions from *"HMSC (High level Message Sequence Charts)"* and *"bMSC (basic Message Sequence Charts)"* [1] specifications of the system [11, 5]. These kinds of specifications are common practice in real-scale contexts thus CCS can merely be regarded as an internal to the framework specification language. Since HMSC and bMSC specifications model finite-state behaviors of a system we will use finite-state CCS:

Definition 1. Connector Free Architecture (CFA). $CFA \equiv (C_1 \mid C_2 \mid ... \mid C_n) \backslash \bigcup_{i=1}^{n} Act_i$ *where for all* $i = 1, .., n$, Act_i *is the actions set of the CCS process* C_i.

Definition 2. Connector Based Architecture (CBA). $CBA \equiv (C_1[f_1] \mid C_2[f_2] \mid ... \mid C_n[f_n] \mid K) \backslash \bigcup_{i=1}^{n} Act_i[f_i]$ *where for all* $i = 1, .., n$, Act_i *is the actions set of the CCS process* C_i *and* f_i *is a relabelling functions such that* $f_i(\alpha) = \alpha_i$ *for all* $\alpha \in Act_i$ *and* K *is the CSS process representing the connector.*

4. APPROACH DESCRIPTION

In this section we recall our approach to the automatic component-based systems generation discussed in our precedent works [7, 5, 11, 8, 6]. We briefly report the aspects of the approach which are important to show, in Section 5, its compositionality with respect to system evolution. The problem treated by our approach was informally phrased as follows: *given a CFA system T for a set of black-box interacting components and a set of properties P automatically derive the corresponding CBA system V which satisfies every property in P.* We assume that a high-level description of the system to be assembled was provided. Referring to Definition 1, we assume that for each component a description of its behavior as finite-state CCS term is provided (i.e. LTS *Labelled Transitions System*). Moreover we assume that a specification of the behavioral properties to be checked exists. Our method proceeds in three steps as illustrated in Figure 2.

The first step builds a connector following the CBA style

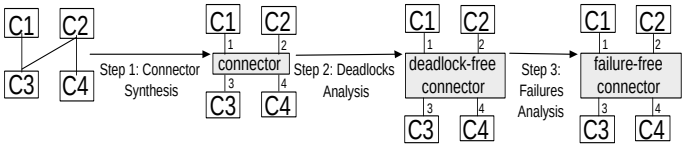

Figure 2: 3 step method

constraints. The second step performs the deadlocks detection and recovery process. Finally, the third step performs the check of the specified behavioral properties against the deadlock-free connector model and then synthesizes a properties-satisfying connector model. For the aims of this paper we do not report in detail the steps of our approach. For a detailed description we refer to [6].

4.1 First step: Connector Synthesis

The first step of our method (see Figure 2) starts with a CFA system and produces the equivalent CBA system. It is worthwhile noticing that this can always be done [8]. We proceed as follows:

i) for each CCS component specification in the CFA system we derive the corresponding AC-Graph. AC-Graphs model components behavior in terms of interactions with the external environment. AC-Graph carry on information on both labels and states. In Figure 3 we show the AC-Graphs of the CFA system of our working example.

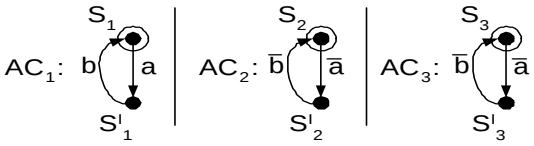

Figure 3: AC-Graphs of the example

In the example, we are considering a client-server (i.e. single-layer) system with two clients (AC_2, AC_3) and one server (AC_1). The server component exports two methods (a and b). It assumes that an its client requires first the method a and then the method b. The two clients have the same behavior. Each client sequentially performs first a request of a and then a request of b.

ii) We derive from AC-Graph the requirements on its environment that guarantee deadlock freedom. Referring to Definition 1, the environment of a component C_i is represented by the set of components C_j ($j \neq i$) in parallel. A component will not block if its environment can always provide the actions it requires for changing state. This is represented as AS-Graphs (Figure 4).

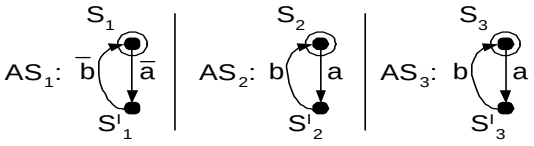

Figure 4: AS-Graphs of the example

Now if we consider Definition 2, the environment of a component can only be represented by connectors, EX-Graph represents the behavior that the component expects from the connectors (Figure 5).

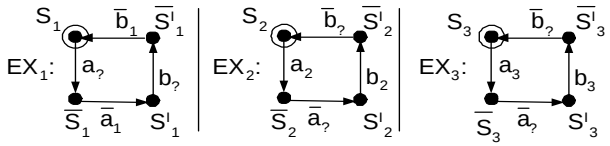

Figure 5: EX-Graphs of the example

iii) Each EX-Graph represents a partial view of the connector behavior, we derive the connector global behavior through an EX-Graphs unification algorithm [8, 6]. In Figure 6 we show the connector graph for our example. The $i-th$ generated node of the connector graph is annotated as K_i and its label is reported in the figure. The resulting CBA system is built as defined in Definition 2.

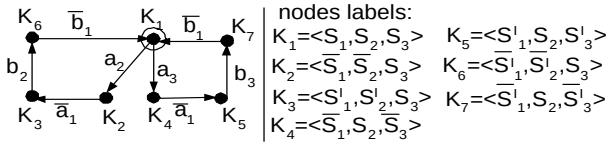

Figure 6: Connector graph of the example

In [8] we have proved that the CBA-system obtained by the connector synthesis process is equivalent to the corresponding CFA-system.

4.2 Second step: Deadlocks recovery

The second step concerns the deadlock freeness analysis, which is performed on the CBA system. Depending on the deadlock type we can operate on the connector in order to obtain a deadlock-free equivalent system. In [8], we have proved that if a *wrong coordination* deadlock is possible, then this results in a precise connector behavior that is detectable by observing the connector graph. To fix this problem it is enough to prune all the finite branches of the connector transition graph. The pruned connector preserves all the correct (with respect to deadlock freeness) behaviors of the CFA-system. In our example (Figure 6), the connector graph does not contain wrong coordination deadlocks. After the elimination of the wrong coordination deadlocks others deadlocks can still occur (i.e. *wrong components assumptions* deadlocks [8]). This is possible because these deadlocks depend on an internal incorrect behavior of the component (e.g. buffer size), thus they do not result in a connector behavior. It is worthwhile recalling that we are dealing with black-box components, whose only known behavior concerns the interactions with the others components into the system. For this reason we have to perform another set of controls on the CBA-system; we check if for every component the connector can simulate the environment's expected behavior from the component (i.e. component's AS-Graph). This check is performed by following a suitable notion of *stuttering* equivalence [10]. In [8] we have proved that if this check is verified then there is no deadlock in the

system. It is easy to perform this check on our working example proving that the connector has no wrong components assumptions deadlocks.

4.3 Third step: Behavioral properties enforcing

The properties we want to enforce are related to behaviors of the CFA system. Behaviors that do not satisfy the specified properties represent behavioral failures of the CFA system. Since the synthesized CBA system is equivalent to the CFA system, these behavioral failures are also related to precise behaviors of the CBA system. Analogously to deadlock [8, 7], we can not solve behavioral failures of the CBA system that are not identifiable with precise behaviors of the synthesized connector. A connector behavior is simply a connector graph states sequence. Thus the system's properties we can deal with are behavioral properties that can be specified in terms of the connector execution states. Since in specifying a property we are concerned with the CFA system, we constrain the set of properties that we can treat. Actually *we deal with behavioral properties specified in terms of connector execution states that are tuples only of AC-Graph states.* By referring to the usual model checking approach [2] we specify every behavioral property through a temporal logic formalism. We choose *LTL* [2] (*Linerar-time Temporal Logic*).

The semantics of a LTL formula is defined with respect to a model represented by a Kripke structure [2]. We consider as Kripke structure corresponding to the connector graph K a connector model KS_K that represents the Kripke structure of K. Let P be the property specification (i.e. a LTL formula), we can translate P in the corresponding Büchi Automaton [2] B_P. Referring to our example we consider the following behavioral property: $P = G(F(< S'_1, S'_2, S_3 >)) \wedge G(F(< S'_1, S_2, S'_3 >))$; this property is the specification of all CFA system behaviors that guarantee the evolution of all components in the system. The synthesized connector avoids starvation and allows the CBA-system to satisfy this property. To a Kripke structure T corresponds a Büchi Automaton B_T [2]. We can derive B_{KS_K} as the Büchi Automaton corresponding to KS_K. Thus given $B_{KS_K} = (\Sigma, N, \Delta, \{s\}, N)$ and
$B_P = (\Sigma, S, \Gamma, \{v\}, F)$ the method performs the following property enforcing procedure in order to synthesize a deadlock-free connector graph that satisfies P:

1. *build the automaton* $B^{K,P}_{intersection}$ *that accepts* $L(B_{KS_K}) \cap L(B_P)$*;*
2. *if* $B^{K,P}_{intersection}$ *is not empty return* $B^{K,P}_{intersection}$ *as the Büchi Automaton corresponding to the P-satisfying execution paths of K.*

If $B^{K,P}_{intersection}$ is empty it means that for all possible executions of the CBA system the property is violated. In this case it is impossible to assemble the components through a property-satisfying connector synthesized with our approach. Finally our method derives from $B^{K,P}_{intersection}$ the corresponding connector graph. Then, to terminate the construction of the failures-free connector graph, the method prunes the possible branches terminating with stop nodes. In Figure 7 we show the failures-free connector graph for our example. By visiting this graph and by exploiting the information stored in the states we can derive the code that implements the property-satisfying deadlock-free connector (i.e. the assembly code) analogously to what done for deadlock free connectors [7]. In [6] we have proved the correctness of the property enforcing procedure. That is the CBA-system based on the property-satisfying deadlock-free connector preserves all the property-satisfying behaviors of the corresponding deadlock-free CFA-system.

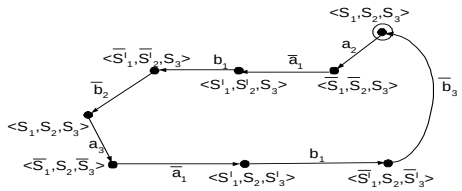

Figure 7: Failures-free connector graph of the example

5. COMPOSITIONAL CONNECTORS SYNTHESIS

Let us suppose that we have to derive the assembly code for a set of components constituting a CFA system S. To do this we want to reuse a connector K already synthesized for a *"reduced"* version of S. With the term *"reduced"* we mean a version of S made of a subset of the set of components constituting it. Referring to Definitions 1 and 2, the problem we want to treat can be formulated as follows:

- *let* $CBA \equiv (C_1[f_1] \mid C_2[f_2] \mid ... \mid C_n[f_n] \mid K) \backslash \bigcup_{i=1}^{n} Act_i[f_i]$ *be the CBA system made of* $C_1, .., C_n$ *components;*
- *let* $CFA' \equiv (C_1 \mid ... \mid C_n \mid C_{n+1} \mid ... \mid C_{n+m}) \backslash (\bigcup_{i=1}^{n} Act_i \cup \bigcup_{j=n+1}^{n+m} Act_j)$ *be the CFA system made of* $C_1, .., C_n, C_{n+1}, .., C_{n+m}$ *components;*

define a function F in a such way that we automatically derive the CBA system CBA′ corresponding to CFA′, where:

- $CBA' \equiv (C_1[f_1] \mid ... \mid C_n[f_n] \mid C_{n+1}[f_{n+1}] \mid ... \mid C_{n+m}[f_{n+m}] \mid K') \backslash (\bigcup_{i=1}^{n} Act_i[f_i] \cup \bigcup_{j=n+1}^{n+m} Act_j)$*;*
- $K' = F(K, C_1, .., C_n, C_{n+1}, .., C_{n+m})$.

In Figure 8 we show an instance of the problem we want to treat.

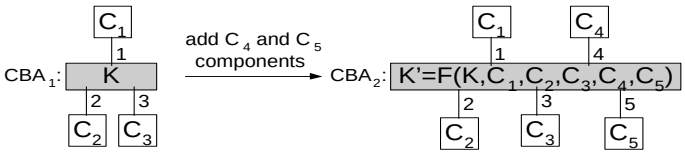

Figure 8: Compositional connector synthesis

The function F is defined as a compositional version of the EX-Graphs unification algorithm defined in [8, 6]. Before

formalizing F, by continuing the example of Section 4 we show how to perform the compositional EX-Graphs unification algorithm. We consider CBA_1, C_4 and C_5 in Figure 8. In Figure 9 we show the AC-Graphs of the components in CBA_1 and the property-satisfying deadlock-free connector graph K for CBA_1 (Figure 7). We are assuming the behavioral property P specified in Section 4.3. We also show the AC-Graphs of the components in CBA_2 and the connector graph $K' = F(K, C_1, C_2, C_3, C_4, C_5)$ for CBA_2.

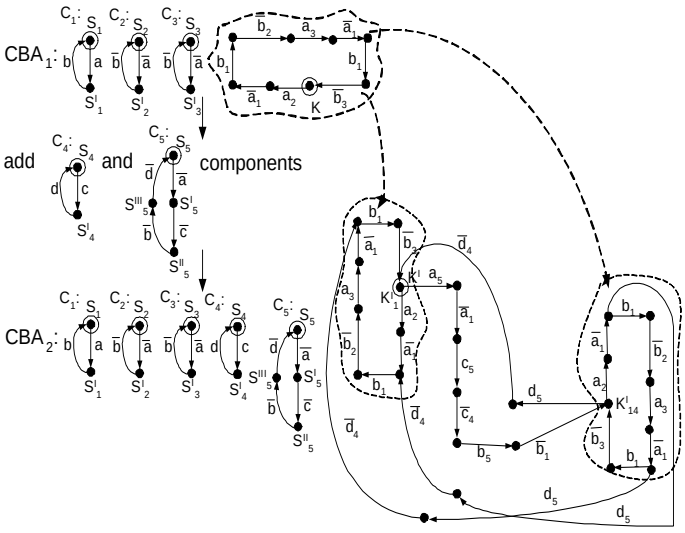

Figure 9: $K' = F(K, C_1, C_2, C_3, C_4, C_5)$

In a client-server setting, this example describes a CBA system (CBA_1) made of the connector (server) component K assembling the server C_1 (contained in K) and the two clients C_2 and C_3. C_1 can sequentially receive two requests of service modelled by input actions a and b respectively. C_2 and C_3 have the same behavior. They first perform a request of service toward C_1 modelled by output action $\overline{a}$ and then perform the request modelled by $\overline{b}$. The connector component implements the P-satisfying deadlock-free routing policy of the two clients requests toward the server contained in the connector itself. Let us suppose we want to add functionalities to CBA_1. For instance we add a new server C_4 and a new client C_5 to CBA_1. C_4 can sequentially receive two requests of service (c and d actions), C_5 sequentially performs four requests ($\overline{a}$, $\overline{c}$, $\overline{b}$, $\overline{d}$); two requests ($\overline{a}$ and $\overline{b}$) toward the old server C_1 and two ($\overline{c}$ and $\overline{d}$) toward the added server C_4.

To automatically derive the connector K' representing the P-satisfying deadlock-free assembly code for C_1, C_2, C_3, C_4 and C_5 components we can exploit the already generated connector component K in order to reduce the number of iterations of the EX-Graph unification algorithm and to avoid the enforcing of the already validated property (i.e. P). We perform a compositional EX-Graphs unification algorithm. This algorithm defines the function F. The compositional EX-Graphs unification algorithm gets in input a connector graph K and the EX-Graphs EX_i of every component in the system to be assembled. Moreover the algorithm denotes both the EX-Graphs of the components of the old CBA system and the EX-Graphs of the new added components by maintaining them in two EX-Graphs sets: EX_{old} and EX_{new} respectively. Then the algorithm is quite similar to the algorithm defined in [8, 6]. The only difference is in the unification procedure. Informally the unification procedure is performed only for the unifiable pairs $(t_i, v_?)$ where in EX_i $S_i \stackrel{t_i}{\Rightarrow} \overline{S}_i$; in EX_j $S_j \stackrel{v_?}{\Rightarrow} \overline{S}_j$; $i \neq j$; and there exists k with $k = i \vee k = j$ such that $EX_k \in EX_{new}$. For all others $(t_i, v_?)$ unifiable pairs (i.e. the pairs such that in EX_i $S_i \stackrel{t_i}{\Rightarrow} \overline{S}_i$; in EX_j $S_j \stackrel{v_?}{\Rightarrow} \overline{S}_j$; $i \neq j$; and EX_i, $EX_j \in EX_{old}$) the unification procedure directly adds, if they do not already exist, the state transitions of the connector graph K by starting from the current state and by suitable relabelling the states in these transitions. Moreover the compositional unification procedure adds the state transitions of the connector graph K only in correspondence of current states $< S_1, S_2, S_3, S_4, S_5 >$ where S_1, S_2 and S_3 are the initial states of EX_1, EX_2 and EX_3 respectively (i.e. $< S_1, S_2, S_3 >$ is the initial state of K). Instead S_4 and S_5 can be generic states of EX_4 and EX_5 respectively. In Figure 10 we show the resulting connector graph of $K' = F(K, C_1, C_2, C_3, C_4, C_5)$ after the first step of the compositional unification procedure.

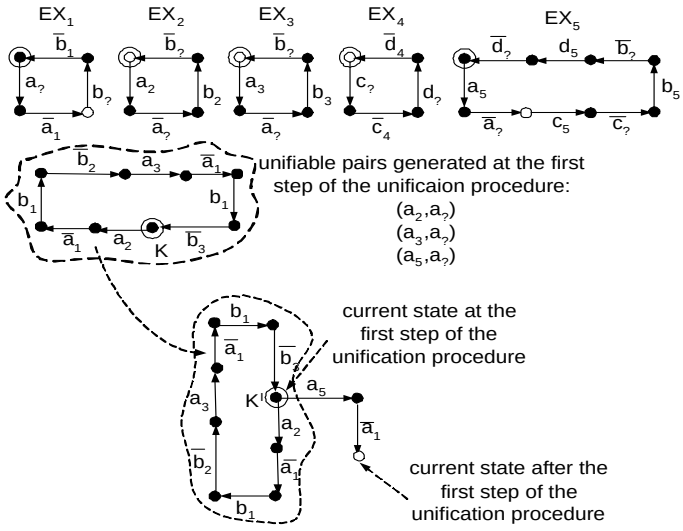

Figure 10: Connector graph of K' after the first step of the compositional unification procedure

The empty EX-Graphs nodes in Figure 10 represent the EX-Graphs current states after the first step of the compositional unification procedure. The unification procedure is performed only for the unifiable pair $(a_5, a_?)$. For the unifiable pairs $(a_2, a_?)$ and $(a_3, a_?)$ the unification procedure simply adds the state transitions in the connector graph K by starting from the current state. The following is the formal definition of the function F:

Definition 3. Compositional EX-Graphs Unification.
Let $C_1, .., C_{n+m}$ be the components in CFA-version of the composed system in such a way that $\{C_1, .., C_h\}$ is the set of null bottom domain components and $\{C_{h+1}, .., C_{n+m}\}$ is the set of null top domain components.
Let $EX_1, .., EX_h, EX_{h+1}, .., EX_{n+m}$ be their corresponding EX-Graphs.

Let $1,..,h,h+1,..,n+m$ be their corresponding communication channels.
Let $S_1,..,S_h,S_h+1,..,S_{n+m}$ be their corresponding current states.
Let K the connector graph already synthesized for the subset $\{C_1,..,C_n\}$ of the set $\{C_1,..,C_n,C_{n+1},..,C_{n+m}\}$; K is the connector for the components in the old CBA-version of the system.
Let EX_{old} the set of EX-Graphs of the components in the old CBA-version of the system; $EX_{old}=\{EX_1,..,EX_n\}$.
Let EX_{new} the set of EX-Graphs of the new added components; $EX_{new}=\{EX_{n+1},..,EX_{n+m}\}$.

At the beginning the current states are the initial states.

1. *Create the actual behavior graph of the connector, with one node (initial state) and no arcs.*
2. *Set as current states of the components $EX-Graphs$ the respective initial states.*
3. *Label the connector initial state with an ordered tuple composed of the initial states of all components (null bottom domain and null top domain). For simplicity of presentation we assume to order them so that the $j-th$ element of the state label corresponds to the current state of the component C_j where $j \in [1,..,h,h+1,..,n+m]$. This state is set as the connector current state.*
4. *Perform the following unification procedure:*
   (a) *Let g be the connector current state. Mark g as visited.*
   (b) *Let $< S_1,..,S_h,S_{h+1},..,S_{n+m} >$ be the state label of g.*
   (c) *Generate the set TER of action_terms and the set VAR of action_variables so that $t_i \in TER$, if in EX_i $S_i \stackrel{t_i}{\Rightarrow} \overline{S}_i$. Similarly $v_? \in VAR$, if $\exists$ j in such a way that in EX_j $S_j \stackrel{v_?}{\Rightarrow} \overline{S}_j$.*
   (d) *For all unifiable pairs $(t_i,\ v_?)$, with $i \neq j$ and $EX_i,\ EX_j \in EX_{old}$, if $g =< S_1,..,S_{n+m} >$ is such that $< S_1,..,S_n >$ is the label of the initial state of K do: if they do not already exist, add the state transitions of K by starting from g and relabel every state $< S_1^k,..,S_n^k >$ of the added K with $< S_1^k,..,S_n^k,S_{n+1},..,S_{n+m} >$.*
   (e) *For all unifiable pairs $(t_i,\ v_?)$, with $i \neq j$ and there exists k with $k=i \vee k=j$ such that $EX_k \in EX_{new}$ do:*
      i. *if $i \in \{1,..,h\}$, $j \in \{h+1,..,n+m\}$ and they do not already exist then create new nodes (in the connector graph) g_i, g_j with state label $< S_1,..,\overline{S}_i,..,S_h,S_{h+1},..,\overline{S}_j,..,S_{n+m} >$ and $< S_1,..,S'_i,..,S_h,S_{h+1},..,S'_j,..,S_{n+m} >$ respectively, where in AS_i $S_i \stackrel{t}{\Rightarrow} S'_i$ and in AS_j $S_j \stackrel{\overline{v}}{\Rightarrow} S'_j$;*
      ii. *if $j \in \{1,..,h\}$, $i \in \{h+1,..,n+m\}$ and they do not already exist then create new nodes (in the connector graph) g_i, g_j with state label $< S_1,..,\overline{S}_j,..,S_h,S_{h+1},..,\overline{S}_i,..,S_{n+m} >$ and $< S_1,..,S'_j,..,S_h,S_{h+1},..,S'_i,..,S_{n+m} >$ respectively, where in AS_i $S_i \stackrel{t}{\Rightarrow} S'_i$ and in AS_j $S_j \stackrel{\overline{v}}{\Rightarrow} S'_j$;*
      iii. *create the arc $(g,\ t_i,\ g_i)$ in the connector graph;*
      iv. *mark g_i as visited;*
      v. *create the arc $(g_i,\ \overline{v}_j,\ g_j)$ in the connector graph.*
   (f) *Perform recursively this procedure on all not marked (as visited) adjacent nodes of current node.*

By Definition 3 is trivially proved the correctness of the compositional connectors synthesis. That is $K' = F(K,C_1,..,C_n,C_{n+1},..,C_{n+m})$ is equal to K' simply obtained by performing the unification algorithm defined in [8, 6] on $C_1,..,C_n,C_{n+1},..,C_{n+m}$ and by enforcing the behavioral property P. It is worthwhile noticing that P is specified in terms of tuples of states of components in CBA_1. When we say that K' satisfies P we mean that K' satisfies a set of behavioral properties deriving from the set of all the behavioral properties that can be derived by considering P in CBA_2. For instance we have $P = G(F(< S'_1,S'_2,S_3 >)) \wedge G(F(< S'_1,S_2,S'_3 >))$. Considering CBA_2 we can see P as the set: $\{P\ :\ P = G(F(< S'_1,S'_2,S_3,X,Y >)) \wedge G(F(< S'_1,S_2,S'_3,X,Y >))$ *for all states* $g =< S_1,S_2,S_3,X,Y >$ *in* K' *in which the compositional unification procedure has applied the step* (d) $\}$. Thus in our example P in CBA_1 corresponds to the set of properties $\{P_1,P_2\}$ in CBA_2 where $P1 = G(F(< S'_1,S'_2,S_3,S_4,S_5 >)) \wedge G(F(< S'_1,S_2,S'_3,S_4,S_5 >))$ and $P2 = G(F(< S'_1,S'_2,S_3,S'_4,S'''_5 >)) \wedge G(F(< S'_1,S_2,S'_3,S'_4,S'''_5 >))$.

6. CONCLUSION AND FUTURE WORKS

In this paper we have have showed the compositional nature of an architectural approach to the automatic component-based systems generation. The approach was already developed in our precedent works [7, 5, 11, 8, 6]. The main contribution of this paper is to show that our approach is compositional with respect to the generation of a new version of the system and to system's behavioral specification. As future works we have to implement the compositional connectors synthesis algorithm, defined in Section 5, in the current version of the framework [7, 5, 11, 8, 6]. Moreover in order to verify the applicability and generality of the approach, we have to apply this compositional technique to real-context case studies.

7. REFERENCES

[1] Itu telecommunication standardisation sector, itu-t reccomendation z.120. message sequence charts. (msc'96). Geneva.

[2] O. G. Edmund M. Clarke, Jr. and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, London, England, 2001.

[3] D. Garlan and D. E. Perry. *Introduction to the Special Issue on Software Architecture, Vol. 21. Num. 4. pp. 269-274*, April 1995.

[4] J. S. I. Crnkovic, H. Schmidt and K. Wallnau. Anatomy of a research project in predictable assembly. *Fifth ICSE Workshop on Component-Based Software Engineering White paper.*

[5] P. Inverardi and M. Tivoli. Correct and automatic assembly of cots components: an architectural approach. *in proceedings of the 5th ICSE Workshop on Component-Based Software Engineering: Benchmarks for Predictable Assembly at 24th ICSE 2002, Orlando, Florida, USA, May 19-20, 2002.*

[6] P. Inverardi and M. Tivoli. Failures-free connector synthesis for correct components assembly. *Submitted for publication. http://www.di.univaq.it/∼tivoli/icalp2003.pdf.*

[7] P. Inverardi and M. Tivoli. Automatic synthesis of deadlock free connectors for com/dcom applications. In *ACM Proceedings of the joint 8th ESEC and 9th FSE, ACM Press*, Vienna, Sep 2001.

[8] P. Inverardi and M. Tivoli. Connectors synthesis for failures-free component based architectures. *Technical Report, University of L'Aquila, Department of Computer Science, http://sahara.di.univaq.it/tech.php?id_tech=7 or http://www.di.univaq.it/∼tivoli/ffsynthesis.pdf*, ITALY, January 2003.

[9] R. Milner. *Communication and Concurrency.* Prentice Hall, New York, 1989.

[10] R. D. Nicola and F. Vaandrager. Three logics for branching bisimulation. *Journal of the ACM*, 42(2):458–487, 1995.

[11] P.Inverardi and M.Tivoli. Automatic failures-free connector synthesis: An example. *Technical Report, published on the Monterey 2002 Workshop Proceedings: Radical Innovations of Software and Systems Engineering in the Future, Universita' Ca' Foscari di Venezia, Dip. di Informatica, Technical Report CS-2002-10*, September 2002.

[12] C. Szyperski. *Component Software. Beyond Object Oriented Programming.* Addison Wesley, Harlow, England, 1998.