Synthesizing Glue Operators from Glue Constraints for the Construction of Component-Based Systems

Simon Bliudze[1] and Joseph Sifakis[2]

[1] CEA, LIST, Boîte Courrier 94, Gif-sur-Yvette, F-91191 France
Simon.Bliudze@cea.fr
[2] VERIMAG, Centre Équation, 2 av de Vignate, 38610, Gières, France
Joseph.Sifakis@imag.fr

Abstract. We study glue operators used in component-based frameworks to obtain systems as the composition of atomic components described as labeled transition systems (LTS). Glue operators map tuples of LTS into LTS. They restrict the behavior of their arguments by performing memoryless coordination. In a previous paper, we have proposed a simple format for SOS rules that captures, in particular, glue operators from known frameworks such as CCS, SCCS, CSP, and BIP.

This paper studies a new way for characterizing glue operators: as boolean *glue constraints* between interactions (sets of ports) and the state of the coordinated components. We provide an SOS format for glue, which allows a natural correspondence between glue operators and glue constraints. This correspondence is used for automated synthesis of glue operators implementing given glue constraints. By focusing on the properties that do not bear computation, we reduce a very hard (and, in general, undecidable) problem of synthesizing controllers to a tractable one. The examples in the paper show that such properties are natural and can be expressed as glue constraints in a straightforward manner. Finally, we compare expressiveness of the proposed formalisms with the glue used in the BIP framework and discuss possible applications.

1 Introduction

A central idea in systems engineering is that complex systems are built by assembling components. Large components are obtained by "gluing" together simpler ones. "Gluing" can be considered as a generalized composition operation on sets of components that allows building complex components from simpler ones.

Various component frameworks exist for developing hardware, software or mixed hardware/software systems. They all focus rather on the way components interact than on their internal behavior. They often use domain specific mechanisms for composing components, each mechanism implying specific approaches for organizing their interaction. For instance, hardware systems are built by using buses implementing usually synchronous multi-party interaction.

S. Apel and E. Jackson (Eds.): SC 2011, LNCS 6708, pp. 51–67, 2011.

Asynchronous message passing is very common in operating systems and middleware. For software the main paradigms are using lock/unlock operations or (blocking) function calls, although domain-specific design languages may rely on different mechanisms, e.g., broadcast for synchronous languages like Esterel and rendezvous in ADA. Such a heterogeneity is a main obstacle for

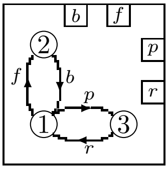
- comparing functionally equivalent designs integrating identical sets of components but using different types of "glue" for their coordination: How to evaluate the merits of different solutions by using theoretical tools and reasoning rather than experimental analysis of their implementations?
- composing systems based on different composition paradigms, often based on different models of computation that cannot be consistently combined.

In previous papers, we have advocated for component frameworks using rather families of composition operators than a single composition operator. This allows mastering complexity in designs and enhanced expressiveness. BIP (Behavior-Interaction-Priority) [1] is such a framework combining two families of composition operators: interactions and priorities. In [2], we formalized the concept of glue operator for behavior coordination by using SOS inference rules in a very simple restriction of the GSOS format [3]. Each operator gl composing behaviors $B_1, \dots, B_n$ is defined by a set of inference rules of the form

$$\frac{\{B_i : q_i \xrightarrow{a_i} q_i'\}_{i\in I} \quad \{q_i = q_i'\}_{i\not\in I} \quad \{B_j : q_j \not\xrightarrow{b_j^k}\}_{\substack{j\in J\\ k\in K_j}}}{gl(B_1, \dots, B_n) : q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'}\,, \tag{1}$$

where a, in the conclusion, is the interaction obtained as the union of the interactions occurring in positive premises.

Example 1 (Mutual exclusion by preemption). Let T_1 and T_2 be two mutually preemptable tasks sharing a single processor for their computations. No interactions other than those needed for the preemption mechanism are possible. Tasks can be modeled by the generic behavior shown to the right. This behavior has three states: 1–the task is running, 2–the task is waiting to begin computation, and 3–the task has been preempted and is waiting to resume computation. The transitions are labeled b, f, p, and r for *begin*, *finish*, *preempt*, and *resume* respectively, and can be synchronized with external events through the corresponding ports of the behavior. Mutual preemption is described by two statements:



1. A running task is preempted, when the other one begins computation.
2. A preempted task resumes computation, when the other one finishes.

The glue operator gl ensuring this behavior in the composition of the two tasks is defined by the inference rules shown in Fig. 1 (for $i, j = 1, 2$ and $i \neq j$).

In [2], we have introduced a notion of expressiveness for component frameworks and have shown that BIP is as expressive as the family of the glue operators specified by (1).

$$\frac{B_i : q_i \xrightarrow{b_i} q_i' \quad B_j : q_j \overset{p_j}{\not\rightarrow}}{gl(B_1,B_2) : q_i q_j \xrightarrow{b_i} q_i' q_j} \qquad \frac{B_i : q_i \xrightarrow{b_i} q_i' \quad B_j : q_j \xrightarrow{p_j} q_j'}{gl(B_1,B_2) : q_i q_j \xrightarrow{b_i p_j} q_i' q_j'}$$

$$\frac{B_i : q_i \xrightarrow{f_i} q_i' \quad B_j : q_j \overset{r_j}{\not\rightarrow}}{gl(B_1,B_2) : q_i q_j \xrightarrow{f_i} q_i' q_j} \qquad \frac{B_i : q_i \xrightarrow{f_i} q_i' \quad B_j : q_j \xrightarrow{r_j} q_j'}{gl(B_1,B_2) : q_i q_j \xrightarrow{f_i r_j} q_i' q_j'}$$

Fig. 1. Inference rules for the mutual exclusion example

This paper studies a new way for characterizing glue operators for behavior coordination. We consider that a glue operator on a set of components is a *glue constraint*. This is a boolean constraint between interactions that can be fired and the state of the coordinated components. The state is characterized by the set of the ports through which interactions are possible. For each port of a component, the constraint has two variables—an activation variable p and a firing variable $\dot{p}$—connected by an additional axiom $\dot{p} \Rightarrow p$. For a given valuation of activation variables corresponding to some state of the interacting components, glue constraints characterize all the possible interactions. An interaction is possible from this state if the valuation setting to *true* only the firing variables corresponding to its ports satisfies the constraint. The axiom $\dot{p} \Rightarrow p$ expresses the fact that a port cannot participate in an interaction unless it is active.

Example 2. The two statements describing preemption in Ex. 1 are formalized by the following constraints (for $i, j = 1, 2$ and $i \neq j$):

- $\dot{p}_i \Rightarrow \dot{b}_j$, meaning that one task can be preempted only when the other one starts computation;
- $\dot{b}_i \wedge p_j \Rightarrow \dot{p}_j$, meaning that when one task begins computation, if the other one is computing (can be preempted) it must be preempted;
- $\dot{r}_i \Rightarrow \dot{f}_j$, meaning that a preempted task can resume only when the other one finishes computation;
- $\dot{f}_i \wedge r_j \Rightarrow \dot{r}_j$, meaning that when one task finishes computation, if the other one is preempted (can resume) it must resume.

In this paper, we define a new variation of the SOS format for glue operators proposed in [2]. In the new format, inference rules operators have three types of premises. The *firing* premises $q_i \xrightarrow{a_i} q_i'$ are the same as positive premises in (1). The two other types—*witness* premises $q_i \uparrow a_i$ and *negative* premises $q_i \not\uparrow a_i$—use a new predicate $\uparrow$. In an atomic behavior B_i, $q_i \uparrow a_i$ is satisfied iff, for every port $p \in a_i$, there is a transition $q_i \xrightarrow{b}$ such that $p \in b$. In other words, every port in a_i is *offered* by some transition in the state q_i of B_i. For a composed behavior B, $q \uparrow a$ signifies that each port in a is offered by some atomic component composing B. As in (1), the conclusion of a rule is labeled by the union of interactions labeling its firing premises, that is neither witness nor negative premises contribute to the resulting transition.

This new format has two advantages. Firstly, it is well adapted to formalizing hierarchical composition of behaviors, as the predicate $\uparrow$ allows to explicitly capture the notion of "atomic" component. Secondly, for each glue constraint there exists an equivalent glue operator defined by rules in the new SOS format. For any such glue operator, it is possible to find an equivalent glue constraint.

This paper is structured as follows. In Sect. 2, we define a new class of glue operators, which we call *universal glue*, as well as glue constraints. In Sect. 3, we show that glue constraints can encode universal glue in a way that allows to explicitly link glue operators to invariant properties of the composed systems. In particular, this allows the synthesis of glue operators enforcing such properties as illustrated in Sect. 4. In Sect. 5, we show that each type of premises in the proposed SOS format is essential for the expressiveness of universal glue. In Sect. 6, we compare this new universal glue with the glue used in BIP. In Sect. 7, we present a design methodology based on the introduced models and transformations. Finally, we discuss the related work in Sect. 8.

2 Modeling Behavior and Glue

2.1 Behavior

Definition 1. *A* labeled transition system *(LTS) is a triple* $(Q, P, \rightarrow)$*, where* Q *is a set of* states*,* P *is a set of* ports*, and* $\rightarrow \subseteq Q \times 2^P \times Q$ *is a set of* transitions*, each labeled by an* interaction*. For* $q, q' \in Q$ *and* $a \in 2^P$*, we write* $q \xrightarrow{a} q'$ *iff* $(q, a, q') \in \rightarrow$*. An interaction* $a \in 2^P$ *is* active *in a state* $q \in Q$ *(denoted* $q \xrightarrow{a}$*), iff there exists* $q' \in Q$ *such that* $q \xrightarrow{a} q'$*. We abbreviate* $q \overset{a}{\not\rightarrow} \overset{def}{=} \neg(q \xrightarrow{a})$*.*

To simplify notation, we write, pq for the interaction $\{p, q\}$.

Definition 2. *A* behavior *is a pair* $B = (S, \uparrow)$ *consisting of an LTS* $S = (Q, P, \rightarrow)$ *and an* offer *predicate* $\uparrow$ *on* $Q \times P$ *such that* $q \uparrow p$ *holds (a port* $p \in P$ *is* offered *in a state* $q \in Q$*) whenever there is a transition from* q *containing* p*, that is* $(\exists a \in 2^P : p \in a \wedge q \xrightarrow{a}) \Rightarrow q \uparrow p$*. The set of ports* P *is the* interface *of* B*.*

The offer predicate extends to interactions: for $a \in 2^P$, $q \uparrow a \overset{def}{=} \bigwedge_{p \in a} q \uparrow p$. For $p \in P$, we have $q \uparrow p = q \uparrow \{p\}$. It is also important to observe that $\uparrow$ and $\rightarrow$ do not coincide: e.g., for a state q and ports p_1, p_2, if the only transitions possible from q are $q \xrightarrow{p_1}$ and $q \xrightarrow{p_2}$, one has $q \uparrow p_1 p_2$, but $q \overset{p_1 p_2}{\not\longrightarrow}$.

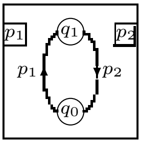
We write $B = (Q, P, \rightarrow, \uparrow)$ for $B = ((Q, P, \rightarrow), \uparrow)$.

Definition 3. *A behavior* $B = (Q, P, \rightarrow, \uparrow)$ *is* atomic *iff,* $\forall q \in Q, p \in P$*,* $(q \uparrow p \Leftrightarrow \exists a \in 2^P : p \in a \wedge q \xrightarrow{a})$*. A behavior is* composed *if it is not atomic.*

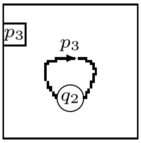
According to this definition, the same transition system S can define an atomic or a composed behavior $B = (S, \uparrow)$, depending on the offer predicate. Thus, the offer predicate allows to capture elements of the components in a composed behavior as illustrated by the following example.

Example 3. Consider the following transition systems:

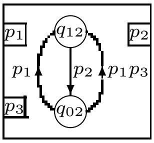
S_1:



S_2:



S_3:



S_3 can be considered as a composition of S_1 and S_2 with an operator that enforces for port p_3 a synchronization with port p_1, i.e. each of p_1 and p_2 can happen alone, but p_3 can happen only together with p_1. Thus, in the state q_{12} of S_3, there is no transition containing p_3, even though p_3 is active in the state 2 of S_2. This can be reflected by considering the behaviors $B_i = (S_i, \uparrow_i)$, for $i = 1, 2, 3$, with $\uparrow_i$ defined by the first three truth tables below.

B_1:

$\uparrow_1$	p_1	p_2
q_0	T	F
q_1	F	T

B_2:

$\uparrow_2$	p_3
q_2	T

B_3:

$\uparrow_3$	p_1	p_2	p_3
q_{02}	T	F	T
q_{12}	F	T	T

B_3':

$\uparrow_3'$	p_1	p_2	p_3
q_{02}	T	F	T
q_{12}	F	T	**F**

Although, there is no interaction a such that $p_3 \in a$ and, in B_3, $q_{12} \xrightarrow{a}$, we have $q_{12} \uparrow p_3$, reflecting the fact that, in the behavior B_2 composing B_3, the port p_3 is active. On the other hand, S_3 can also be considered as "atomic", that is without any explicit information about its structure, by considering a behavior $B_3' = (S_3, \uparrow_3')$ with $\uparrow_3'$ defined by the fourth truth table above.

Note 1. In the rest of the paper, whenever we speak of a set of component behaviors $B_i = (Q_i, P_i, \to, \uparrow)$ with $i \in [1, n]$, we always assume that $\{P_i\}_{i=1}^n$ are pairwise disjoint (i.e. $i \neq j$ implies $P_i \cap P_j = \emptyset$) and $P \overset{def}{=} \bigcup_{i=1}^n P_i$.

Also, to avoid excessive notation, here and in the rest of the paper, we drop the indices on the transition relations $\to$ and offer predicates $\uparrow$, as they can always be unambigously deduced from the state variables, e.g., $q_i \to$ always refers to the transition relation of the corresponding component B_i.

2.2 SOS Characterization of Glue

Structured Operational Semantics (SOS) [4] has been used to define the meaning of programs in terms of Labeled Transition Systems (LTS). A number of SOS formats have been developed, using various syntactic features [5].

In the context of component-based systems, definition of glue only requires the specification of parallel composition operators, as sequential and recursive computation can be represented by individual behaviors.

The SOS rules format below is a modification of the one defined in [2].

Definition 4. *An n-ary* glue operator *gl on a set of interfaces $\{P_i\}_{i=1}^n$ is defined as follows. The application of gl to behaviors $B_i = (Q_i, P_i, \to, \uparrow)$, for $i \in [1, n]$, is a behavior $gl(B_1, \ldots, B_n) = (Q, P, \to, \uparrow)$, with*

- *the set of states $Q = \prod_{i=1}^n Q_i$—the cartesian product of the sets of states Q_i,*
- *the set of ports $P = \bigcup_{i=1}^n P_i$,*
- *the minimal offer predicate $\uparrow$ satisfying, for $i \in [1, n]$, the inference rules*

$$\frac{B_i : q_i \uparrow p}{gl(B_1, \ldots, B_n) : q_1 \ldots q_n \uparrow p}, \qquad (2)$$

- *the minimal transition relation $\rightarrow$ satisfying a set of rules of the form*

$$r = \frac{\{B_i : q_i \xrightarrow{a_i} q_i'\}_{i \in I} \quad \{B_j : q_j \uparrow b_j\}_{j \in J} \quad \{B_k : q_k \not\uparrow c_s \,|\, s \in L_k\}_{k \in K}}{gl(B_1, \dots, B_n) : q_1 \dots q_n \xrightarrow{a} \widetilde{q_1} \dots \widetilde{q_n}} \qquad (3)$$

where $I, J, K \subseteq [1, n]$ and $I \neq \emptyset$; $a = \bigcup_{i \in I} a_i$; and $\widetilde{q_i}$ denotes q_i', for $i \in I$, and q_i, for $i \in [1, n] \setminus I$. In (3), we have three types of premises respectively called firing, witness, *and* negative *premises. Firing and witness premises are collectively called* positive.

The condition $I \neq \emptyset$ means that r has at least one firing premise. Firing premises identify transitions that must actually be taken should the rule be applied; hence there is at most one firing premise per component behavior.

Notice that $q \uparrow b_1 \wedge q \uparrow b_2 = q \uparrow b_1 b_2$, i.e. several witness premises can always be merged into a single more complex one. Hence one witness premise per component behavior is sufficient to define any inference rule.

On the contrary, the conjunction of two negative premises $q \not\uparrow b_1 \wedge q \not\uparrow b_2 = \neg(q \uparrow b_1 \vee q \uparrow b_2)$ cannot be expressed as a primitive expression in terms of the transition relation $\rightarrow$ or offer predicate $\uparrow$. Hence, several negative premises for the same component behavior can be necessary to define an inference rule.

A rule is completely defined by its premises.

We identify the glue operator gl with its set of inference rules (3). A glue operator having no negative premises in any of its rules is called a *positive glue operator.* We call *glue* a set of glue operators and *universal glue* the glue consisting of all the glue operators in the sense of Def. 4. We denote the latter *FWN* (for Firing-Witness-Negative; see also Sect. 5).

2.3 Boolean Characterization of Glue

Let P be a set of ports, we denote $\dot{P} \stackrel{def}{=} \{\dot{p} \,|\, p \in P\}$.

Definition 5. *The* Algebra of Glue Constraints *on P, denoted $\mathcal{GC}(P)$, is the boolean algebra $\mathbb{B}[P, \dot{P}]$ on the set of variables $P \cup \dot{P}$ with an additional axiom $\dot{p} \Rightarrow p$. We call $p \in P$* activation *variables and $\dot{p} \in \dot{P}$* firing *variables.*

Activation variables indicate which ports are offered; firing variables indicate which ports will actually participate in a transition (interaction). Clearly, a port can participate in a transition only if it is offered; hence the axiom $\dot{p} \Rightarrow p$.

Note 2. Below, we overload the symbol P: depending on the context, it will denote the set of ports in a system or the set of the associated boolean variables.

We give the semantics of $\mathcal{GC}(P)$ by associating a glue operator (Def. 4) to each term of the algebra. For behaviors $\{B_i\}_{i=1}^n$, such that $P = \bigcup_{i=1}^n P_i$ where P_i is the set of the ports of B_i, and a glue constraint $\varphi \in \mathcal{GC}(P)$, the composed behavior $\varphi(B_1, \dots, B_n)$ can be described intuitively as follows. For each $i \in [1, n]$, the current state of $q_i \in Q_i$ defines a valuation on the activation variables of φ:

for each variable $p \in P_i$, the valuation is $p = true$ iff $q_i \uparrow p$ (the underlying port is offered in the current state of the behavior). An interaction $a \subseteq P$ defines a valuation on the firing variables $\dot{P}$ by putting $\dot{p} = true$ iff $p \in a$. Finally, a is possible in the composite behavior $\varphi(B_1, \dots, B_n)$ iff the valuation on $P \cup \dot{P}$ defined as above satisfies φ and the constituent interactions of a are possible in the corresponding component behaviors.

To formalize the above intuition, let us consider behaviors $B_i = (Q_i, P_i, \to, \uparrow)$ for $i \in [1, n]$ and a glue constraint $\varphi \in \mathcal{GC}(P)$. Denote $\nu_{B_i}^{q_i}(p) \overset{def}{=} q_i \uparrow p$ the valuation on the activation variables in P_i associated to the state $q_i \in Q_i$ of B_i. For an interaction $a \subseteq P$, denote $\mathbb{I}_a(\dot{p}) \overset{def}{=} (p \in a)$ the corresponding valuation on firing variables. The behavior obtained by composing $\{B_i\}_{i=1}^n$ with φ is defined by $\varphi(B_1, \dots, B_n) = (Q, P, \to, \uparrow)$, with $Q = \prod_{i=1}^n Q_i$, the offer predicate$\uparrow$ defined by (2), and the transition relation $\to$ defined as follows. For any $q_1 \dots q_n \in Q$ and $a \subseteq P$, and for all $q_i' \in Q_i$ such that $q_i \xrightarrow{a \cap P_i} q_i'$ for $i \in [1, n]$, we put $q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n' \overset{def}{\Longleftrightarrow} (\nu_{B_1}^{q_1}, \dots, \nu_{B_n}^{q_n}, \mathbb{I}_a) \models \varphi$.

3 Transformations

3.1 From Glue Operators to Glue Constraints

Recall that a glue operator on a set of interfaces $\{P_i\}_{i=1}^n$ is identified with the set of its defining rules. Thus, we first present the translation into $\mathcal{GC}(P)$ of individual rules. Let r be a rule as in (3). Denoting $A = \bigcup_{i \in I} a_i$, $B = \bigcup_{j \in J} b_j$, we associate to r a formula $\varphi_r \in \mathcal{GC}(P)$ defined by

$$\varphi_r \overset{def}{=} \bigwedge_{p \in A} \dot{p} \ \wedge \bigwedge_{p \in P \setminus A} \overline{\dot{p}} \ \wedge \bigwedge_{p \in B} p \ \wedge \bigwedge_{k \in K} \bigwedge_{s \in L_k} \overline{c_s}\,, \tag{4}$$

where $\overline{c_s} = \bigvee_{p \in c_s} \overline{p}$. A formula $\varphi_{gl} \in \mathcal{GC}(P)$ associated to a glue operator gl is then defined by putting $\varphi_{gl} \overset{def}{=} \bigvee_{r \in gl} \varphi_r$.

Proposition 1. *Let $gl \in FWN$ be defined on interfaces $\{P_i\}_{i=1}^n$ and let $\varphi_{gl} \in \mathcal{GC}(P)$ be the glue constraint formula constructed as above. For any set of behaviors $B_i = (Q_i, P_i, \to, \uparrow)$, with $i \in [1, n]$, holds $gl(B_1, \dots, B_n) = \varphi_{gl}(B_1, \dots, B_n)$.*

3.2 From Glue Constraints to Glue Operators

Let again $\{P_i\}_{i=1}^n$ be a set of interfaces and $P = \bigcup_{i=1}^n P_i$. The transformation from $\mathcal{GC}(P)$ to FWN can be defined by giving, for a glue constraint formula $\varphi \in \mathcal{GC}(P)$, an equivalent operator $gl_\varphi \in FWN$ on $\{P_i\}_{i=1}^n$. In order to do so we have to rewrite φ as a disjunction of formulæ of the form (4).

Let $\varphi \in \mathcal{GC}(P)$ be a glue constraint formula. Observe first that φ has a unique representation in the *firing-full DNF*, i.e. a disjunctive normal form such that all firing variables $\dot{p} \in \dot{P}$ are explicitly present in each monomial. Thus $\varphi = \bigvee_{m \in M} \varphi_m$, with monomials (indexed by the set M) of the form

$$\varphi_m = \bigwedge_{p \in A_m} \dot{p} \wedge \bigwedge_{p \in P \setminus A_m} \overline{\dot{p}} \wedge \bigwedge_{p \in B_m} p \wedge \bigwedge_{i=1}^{n} \bigwedge_{p \in C_m \cap P_i} \overline{p}\,. \tag{5}$$

This formula has the same form as in (4). Denote, for $m \in M$ and $i \in [1,n]$, $a_i^m = A_m \cap P_i$ and $b_j^m = B_m \cap P_j$; $I_m = \{i \in [1,n] \,|\, a_i^m \neq \emptyset\}$ and $J_m = \{j \in [1,n] \,|\, b_j^m \neq \emptyset\}$. The glue operator gl_φ corresponding to the formula φ is defined by the set of inference rules, containing, for each $m \in M$, the rule (6), where $\widetilde{q_i}$ denotes q_i', for $i \in I_m$, and q_i, for $i \in [1,n] \setminus I_m$.

$$\frac{\{B_i : q_i \xrightarrow{a_i^m} q_i'\}_{i \in I_m} \quad \{B_j : q_j \uparrow b_j^m\}_{j \in J_m} \quad \{B_k : q_k \not\uparrow p \,|\, p \in C_m \cap P_k\}_{k=1}^n}{gl_\varphi(B_1, \dots, B_n) : q_1 \dots q_n \xrightarrow{A_m} \widetilde{q_1} \dots \widetilde{q_n}} \tag{6}$$

Example 4. Let $P_1 = \{p_1, p_2\}$ and $P_2 = \{p_3, p_4, p_5\}$, and consider an operator gl on these two interfaces that allows only transitions labelled p_1 (i.e. $\dot{p}_1 \overline{\dot{p}_2}\,\overline{\dot{p}_3}\,\overline{\dot{p}_4}\,\overline{\dot{p}_5}$) and, moreover, only when either $p_2 p_3$ is not active or neither is p_4 nor p_5 (i.e. $\dot{p}_1 \Rightarrow \overline{p_2 p_3} \vee \overline{p_4}\,\overline{p_5}$). Taking the firing-full DNF of the conjunction, we obtain $\dot{p}_1 \overline{\dot{p}_2}\,\overline{\dot{p}_3}\,\overline{\dot{p}_4}\,\overline{\dot{p}_5} \wedge (\dot{p}_1 \Rightarrow \overline{p_2 p_3} \vee \overline{p_4}\,\overline{p_5}) = \dot{p}_1 \overline{\dot{p}_2}\,\overline{\dot{p}_3}\,\overline{\dot{p}_4}\,\overline{\dot{p}_5} \wedge (\overline{p_2 p_3} \vee \overline{p_4}\,\overline{p_5}) = \dot{p}_1 \overline{\dot{p}_2}\,\overline{\dot{p}_3}\,\overline{\dot{p}_4}\,\overline{\dot{p}_5}\,\overline{p_2} \vee \dot{p}_1 \overline{\dot{p}_2}\,\overline{\dot{p}_3}\,\overline{\dot{p}_4}\,\overline{\dot{p}_5}\,\overline{p_3} \vee \dot{p}_1 \overline{\dot{p}_2}\,\overline{\dot{p}_3}\,\overline{\dot{p}_4}\,\overline{\dot{p}_5}\,\overline{p_4}\,\overline{p_5}$. The inference rules for the glue operator gl are shown below:

$$\frac{B_1 : q_1 \xrightarrow{p_1} q_1' \quad B_1 : q_1 \not\uparrow p_2}{gl(B_1, B_2) : q_1 q_2 \xrightarrow{p_1} q_1' q_2} \qquad \frac{B_1 : q_1 \xrightarrow{p_1} q_1' \quad B_1 : q_1 \not\uparrow p_3}{gl(B_1, B_2) : q_1 q_2 \xrightarrow{p_1} q_1' q_2}$$

$$\frac{B_1 : q_1 \xrightarrow{p_1} q_1' \quad B_2 : q_2 \not\uparrow p_4 \quad B_2 : q_2 \not\uparrow p_5}{gl(B_1, B_2) : q_1 q_2 \xrightarrow{p_1} q_1' q_2}$$

Lemma 1. *Let $\varphi \in \mathcal{GC}(P)$ be a glue constraint, $gl_\varphi \in FWN$ constructed as above, and $\varphi_{gl_\varphi} \in \mathcal{GC}(P)$ constructed as in Sect. 3.1 to gl_φ. Then $\varphi_{gl_\varphi} = \varphi$.*

Proposition 2. *Let $\{P_i\}_{i=1}^n$ be a set of interfaces, $\varphi \in \mathcal{GC}(P)$ a constraint formula, and $gl_\varphi \in FWN$ constructed as above. For any set of behaviors $B_i = (Q_i, P_i, \rightarrow, \uparrow)$, with $i \in [1,n]$, we have $\varphi(B_1, \dots, B_n) = gl_\varphi(B_1, \dots, B_n)$.*

4 Synthesis of Glue: A Rescue Robot Example

The following example was inspired by the hill-climbing robot discussed in [6]. Consider a robot R confined to a square paved region (Fig. 2(a)) and consisting of four modules: an engine E, a sensor S, a transmitter T, and a navigation system N. The engine can perform two actions: advance the robot one step forward and rotate 90°, controlled respectively through ports a (*advance*) and r (*rotate*). The sensor can measure the temperature in front of the robot and signal whether it is above a given threshold. The action of measuring is controlled through the port m (*measure*), whereas the produced signal can be observed on the port h (*hot*).

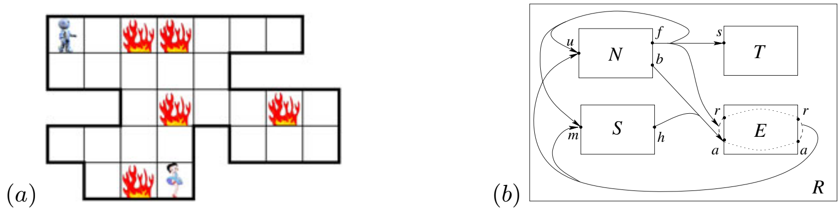


Fig. 2. (a) A rescue robot confined to a square paved region saving Betty Boop from fire; (b) architectural view of the robot.

The transmitter can send the robot's coordinates to the mission control. This action is controlled through the port s (*send*). Finally, the navigation system detects whether the robot is facing the border of the region and whether it has reached the objective. The navigation system data can be updated by an action controlled through the port u (*update*), the proximity of the border is signalled through the port b (*border*) and the objective through the port f (*finish*). Thus, the four interfaces are $P_E = \{a, r\}$, $P_S = \{m, h\}$, $P_T = \{s\}$, and $P_N = \{u, b, f\}$.

We want to synthesize the glue ensuring the following properties, which we encode by glue constraints. Notice, that the synthesized glue only has to ensure the safety of the robot and not that it reaches an objective.

1. The robot must not advance and rotate at the same time: $\overline{\dot a\,\dot r}$;
2. The robot must not leave the region: $b \Rightarrow \overline{\dot a}$;
3. The robot must not drive into hot areas: $h \Rightarrow \overline{\dot a}$;
4. If the robot reaches the objective, it must stop and transmit its coordinates: $f \Rightarrow \overline{\dot a}\,\overline{\dot r}\,\overline{\dot u}\,\overline{\dot m}\,\dot s$;
5. The robot must only transmit its coordinates when it reaches the objective: $\dot s \Rightarrow f$;
6. Every time the robot moves (advances or rotates) the navigation system and the sensor must update their data: $\dot a \vee \dot r \Rightarrow \dot u\,\dot m$.

The architecture of the robot is illustrated in Fig. 2(b), with the controllable ports shown on the left-hand side of each component and the controlling ones on the right-hand side. The ports a and r of the engine are both controllable and controlling, so we duplicate them. The arrows indicate the control influence given by the constraints above.

In order to compute the required glue, we take the conjunction of the six constraints above and the *progress* constraint $(\dot a \vee \dot r \vee \dot u \vee \dot m \vee \dot s) \wedge \overline{\dot h}\,\overline{\dot b}\,\overline{\dot f}$—stating that some controllable action (a, r, u, m, or s) must be taken, but not the non-controllable actions h, b, and f—and apply the procedure described in Sect. 3.2. We rewrite the obtained glue constraint formula in the firing-full DNF: $\overline{\dot a\,\dot r} \wedge (b \Rightarrow \overline{\dot a}) \wedge (h \Rightarrow \overline{\dot a}) \wedge (f \Rightarrow \overline{\dot a}\,\overline{\dot r}\,\overline{\dot u}\,\overline{\dot m}\,\dot s) \wedge (\dot s \Rightarrow f) \wedge (\dot a \vee \dot r \Rightarrow \dot u\,\dot m) \wedge (\dot a \vee \dot r \vee \dot u \vee \dot m \vee \dot s) \wedge \overline{\dot h}\,\overline{\dot b}\,\overline{\dot f} = (\overline{\dot a}\,\overline{\dot r}\,\overline{\dot u}\,\overline{\dot m}\,\dot s\, f \vee \overline{f}\,\overline{\dot s}\,\overline{\dot a}\,\overline{\dot r}\,\dot u\,\overline{\dot m} \vee \overline{f}\,\overline{\dot s}\,\overline{\dot a}\,\overline{\dot r}\,\dot u\,\dot m \vee \overline{f}\,\overline{\dot s}\,\overline{\dot a}\,\overline{\dot r}\,\overline{\dot u}\,\dot m \vee \overline{\dot a}\,\dot r\,\overline{f}\,\overline{\dot s}\,\dot u\,\dot m \vee \dot a\,\overline{\dot r}\,\overline{b}\,\overline{h}\,\overline{f}\,\overline{\dot s}\,\dot u\,\dot m) \wedge \overline{\dot h}\,\overline{\dot b}\,\overline{\dot f}$ Positive parts of all the monomials in this formula are distinct. Therefore no further transformations are necessary, and we directly obtain the following rules defining the required glue operator (we drop the component names, as they are clear from the context):

$$\frac{q_n \xrightarrow{u} q_n' \qquad q_n \not\uparrow f}{q_e q_s q_n q_t \xrightarrow{u} q_e q_s q_n' q_t}\,, \quad \frac{q_s \xrightarrow{m} q_s' \quad q_n \xrightarrow{u} q_n' \quad q_n \not\uparrow f}{q_e q_s q_n q_t \xrightarrow{mu} q_e q_s' q_n' q_t}\,, \quad \frac{q_s \xrightarrow{m} q_s' \qquad q_n \not\uparrow f}{q_e q_s q_n q_t \xrightarrow{m} q_e q_s' q_n q_t}\,,$$

$$\frac{q_t \xrightarrow{s} q_t' \qquad q_n \uparrow f}{q_e q_s q_n q_t \xrightarrow{s} q_e q_s q_n q_t'}\,, \quad \frac{q_e \xrightarrow{r} q_e' \quad q_s \xrightarrow{m} q_s' \quad q_n \xrightarrow{u} q_n' \quad q_n \not\uparrow f}{q_e q_s q_n q_t \xrightarrow{rmu} q_e' q_s' q_n' q_t}\,,$$

$$\frac{q_e \xrightarrow{a} q_e' \quad q_s \xrightarrow{m} q_s' \quad q_n \xrightarrow{u} q_n' \quad q_s \not\uparrow h \quad q_n \not\uparrow b \quad q_n \not\uparrow f}{q_e q_s q_n q_t \xrightarrow{amu} q_e' q_s' q_n' q_t}\,.$$

It is important to observe here that this glue operator ensures the required safety properties independently of the specific implementation of the behaviors of the atomic components (Engine, Sensor, Navigation system and Transmitter), as long as this implementation respects the specified interfaces.

5 The Glue Expressiveness Hierarchy

The rule format for defining glue operators that we introduced in Sect. 2.2 allows three types of premises: firing, witness, and negative. In this section, we use the notion of glue expressiveness [2] to show that all three types are essential for the expressiveness of the glue that can be defined with these rules. To do so, we compare the following four classes of glue operators:

F: the class of glue operators obtained by using rules with only firing premises (of the form $q \xrightarrow{a} q'$). Classical composition operators, e.g., parallel composition in CCS or CSP belong to this class;
FW: the class of glue operators obtained by using rules with only positive premises (firing premises, as above, and witness premises of the form $q \uparrow a$);
FN: the class of glue operators obtained by using rules with only firing and negative (of the form $q \not\uparrow a$) premises;
FWN: the universal glue obtained by using rules with all three types of premises.

Let $\mathcal{B}$ be a set of behaviors with a fixed equivalence relation $\mathcal{R} \subseteq \mathcal{B} \times \mathcal{B}$. A glue is a set G of operators on $\mathcal{B}$. We denote by $\mathcal{G}lue$ the set of all glues on $\mathcal{B}$. We denote $G^{(n)} \subseteq G$ the set of all n-ary operators in G. Thus, $G = \bigcup_{n \geq 1} G^{(n)}$.

To determine whether one glue is more expressive than another, we compare their respective sets of behaviors *composable* from the same *atomic* ones. This consists in exhibiting for each operator of one glue an equivalent operator in the other one. We only consider *strong expressiveness* [2], i.e. , where the exhibited glue operator must be applied to the same set of behaviors as the original one.

Although the results we present can also be extended to *weak expressiveness* [2], where the exhibited glue operator must be applied to the same set of behaviors as the original one with an addition of some fixed set of *coordination behaviors*, we do not present these extensions to avoid overcharging the paper.

Definition 6. *For a given set $\mathcal{B}$ and an equivalence $\mathcal{R}$ on $\mathcal{B}$, the* expressiveness preorder $\preccurlyeq\, \subseteq \mathcal{G}lue \times \mathcal{G}lue$ *w.r.t. $\mathcal{R}$ is defined by putting, for G_1, $G_2 \in \mathcal{G}lue$, $G_1 \preccurlyeq G_2$ if, for any $n \geq 1$ and $B_1, \dots, B_n \in \mathcal{B}$, holds $\forall gl_1 \in G_1^{(n)}\ \exists gl_2 \in G_2^{(n)}$: $gl_1(B_1, \dots, B_n)\, \mathcal{R}\, gl_2(B_1, \dots, B_n)$.*

We consider the partial order induced by $\preccurlyeq$, that is we say that G_1 is *less expressive* than G_2 if $G_1 \preccurlyeq G_2$ and $G_2 \not\preccurlyeq G_1$.

Definition 7. *Let $B_1 = (Q_1, P_1, \rightarrow, \uparrow)$, $B_2 = (Q_2, P_2, \rightarrow, \uparrow)$. A binary relation $\sqsubseteq\, \subseteq Q_1 \times Q_2$ is a* simulation *iff, for all $q_1 \sqsubseteq q_2$ and $a \subseteq P$, $q_1 \uparrow a$ implies $q_2 \uparrow a$ and $q_1 \xrightarrow{a} q_1'$ implies $q_2 \xrightarrow{a} q_2'$, for some $q_2' \in Q_2$ such that $q_1' \sqsubseteq q_2'$.*

We write $B_1 \sqsubseteq B_2$ if there exists a simulation relating each state of B_1 to some state of B_2. $\sqsubseteq$ is the simulation preorder *on behaviors. The relation $\simeq\, = \,\sqsubseteq \cap \sqsubseteq^{-1}$ is the* simulation equivalence *on behaviors.*

Proposition 3. *With respect to the simulation equivalence, F is less expressive than FW and FN, which are both less expressive than FWN.*

6 Glue Constraints, FWN, and BIP

BIP [1,7] is a component framework for constructing systems by superposing three layers of modeling: Behavior, Interaction, and Priorities. The lower layer consists of a set of atomic components modeled by transition systems. The second layer models interactions between components. Priorities are used to enforce scheduling policies applied to interactions of the second layer. Below, we provide a succinct formalization of the BIP component model.

Contrary to Def. 2, a behavior in BIP is an LTS (cf. Def. 1). Let $B_i = (Q_i, P_i, \rightarrow)$, for $i \in [1, n]$, be a set of transition systems and $P = \bigcup_{i=1}^n P_i$. The *composition* of $\{B_i\}_{i=1}^n$, parameterized by a set of interactions $\gamma \subseteq 2^P$, is the transition system $\gamma(B_1, \dots, B_n) = (Q, P, \rightarrow)$, where $Q = \prod_{i=1}^n Q_i$ and $\rightarrow$ is the minimal relation defined by the following set of rules:

$$\left\{ \frac{\{B_i : q_i \xrightarrow{a \cap P_i} q_i' \,|\, i \in I_a\} \quad \{q_i = q_i' \,|\, i \not\in I_a\}}{\gamma(B_1, \dots, B_n) : q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'} \;\middle|\; a \in \gamma \right\}, \tag{7}$$

where, for $a \in \gamma$, $I_a = \{i \in [1, n] \,|\, a \cap P_i \neq \emptyset\}$. We call γ an *interaction model.*

Given a transition system $\gamma(B_1, \dots, B_n)$, a *priority model* π is a strict partial order on 2^P. For $a, a' \in 2^P$, we write $a \prec a'$ iff $(a, a') \in \pi$, meaning that interaction a has less priority than interaction a'. The system obtained by applying the priority model π to $\gamma(B_1, \dots, B_n)$ is the transition system $\pi\gamma(B_1, \dots, B_n) = (Q, P, \rightarrow)$, where Q and P are the same as above, while $\rightarrow$ is the minimal set of transitions satisfying the set of inference rules

$$\pi\gamma = \left\{ \frac{\gamma(B_1, \dots, B_n) : q \xrightarrow{a} q' \quad \{\{a'\}(B_1, \dots, B_n) : q \not\xrightarrow{a'} \,|\, a \prec a'\}}{\pi\gamma(B_1, \dots, B_n) : q \xrightarrow{a} q'} \;\middle|\; a \in \gamma \right\}$$

Intuitively, the premises of a rule for $a \in \gamma$ mean that, in the state $q \in Q$, a is possible in $\gamma(B_1, \ldots, B_n)$, and, for each interaction a' having higher priority than a, there is at least one component B_i such that the constituent interaction $a' \cap P_i$ is not possible in B_i. It is not required that a' belong to γ. In [2], we have shown that the glue of BIP is as expressive as the most general glue defined by inference rules of the form (1).

Example 5. The glue operator for the mutual preemption (Ex. 1) is expressed in BIP by considering the interaction model $\gamma = \{b_i, f_i, b_i p_j, f_i r_j | i, j = 1, 2, i \neq j\}$ and a priority model given by $b_i \prec b_i p_j$ and $f_i \prec f_i r_j$, for $i, j = 1, 2$ and $i \neq j$.

Proposition 4. *With respect to the simulation equivalence, FN is less expressive than the glue in BIP; FWN and the glue in BIP are not comparable.*

An important advantage of BIP is that it allows mastering the complexity of designs by developers: separating glue into two layers corresponding to interactions and priorities drastically simplifies readability of specifications. By redefining the semantics of priority models in terms of the predicate $\uparrow$, it can be shown that this separation can be achieved for the glue operators of Def. 4.

Separation of the coordination level defined by the glue operators into interactions and priorities allows the application of our previous results presented in [7,8]. By comparing (1) and (3), one can observe that both Interaction layer of BIP and positive glue operators of this paper are uniquely characterized by the *interactions* appearing in their premises. For the former, interactions are subsets of 2^P; for the latter, they are subsets of $2^{P \cup \dot{P}}$. The sets P and $\dot{P}$ being disjoint, all the results from the two papers cited above can be applied to connectors of the algebra $\mathcal{AC}(P \cup \dot{P})$. This allows considering structured hierarchical connectors instead of exponential sets of interactions (in BIP) or rules (for glue operators of this paper). Connectors are well suited for compositional design and can be manipulated by symbolic techniques [7]. Furthermore, connectors can be synthesized directly from *causal rules*, which are boolean constraints (implications) on port variables similar to the constraints found in the examples of Sect. 4.

7 Design Methodology

Synthesis of reactive systems has been initiated by Pnueli and Rosner. They have shown that synthesis of *distributed* systems is hard and, sometimes, undecidable [9]. Lustig and Vardi have obtained similar hardness results for synthesis, where reusable components must be selected from component *libraries* [10].

This paper lays ground to a less ambitious, but more tractable design methodology relying on the observation that one of the main difficulties of systems design resides in the concurrent nature of modern software (particularly due to the state space explosion). This methodology can be summarized by the following steps:

1. Choice of the functionalities to be realized by sequential atomic components. This step *does not involve* specifying coordination among these atomic components. It is usually driven by the functional requirements (e.g., functionalities that must be provided by the system) or target platform hardware

specifications. Tools and techniques needed for this design step are, in general, well mastered by software engineers.

2. Independent design of sequential atomic components of the system. As mentioned above, development of sequential programs is much less complex than that of concurrent ones and can also be successfully realized by engineers.
3. Specification of state safety properties to be satisfied by the system. The general case complexity of this design step has yet to be investigated. All the properties that we have encountered are obtained as *causal rules* in a manner similar to that of the example in Sect. 4 (cf. [8]).
4. Automatic glue operator and connector synthesis. This implies that the underlying state safety properties are satisfied *by construction.*

For safety properties other than state properties, compositional deadlock analysis can be performed by adding (sequential) observer components to the synthesized model.

8 Related Work

A number of paradigms for unifying interaction in heterogeneous systems have been studied [12,13,14]. In these works, unification is achieved by reduction to a common low-level semantic model. Coordination mechanisms and their properties are not studied independently of behavior. Our approach is closest to that of [15], where an algebra of connectors is developed that allows one to construct *stateless* connectors from a number of basic ones.

This paper combines several notions that already exist in the literature. We use *boolean constraints* expressed in terms of *activation and firing variables* in order to *synthesize* glue operators expressed in terms of *SOS rules.*

In concurrency, the term *constraints* appears primarily in connection with the constraint automata [16,17], where constraints are used to restrict transition non-determinism. These automata are now widely referred to as having guarded transitions. We speak of *interaction constraints* that characterize mechanisms used in architectures such as connectors, channels, synchronization primitives and result from the composition of actions [18]. Architectural mechanisms are used to constrain the interaction among parallel communicating components.

Few authors have considered the approach by constraints in this context, e.g., [19]. The approach used in [20] is close to the one we adopted in our previous paper [8]. The important difference is that, following the separation of concerns principle, we distinguish coordination and data flow among components.

Separating the coordination layer allows to express coordination constraints as boolean formulæ. In [8], we consider *causal rules*, which are boolean constraints on port variables. Causal rules are used to synthesize connectors to describe interactions among components in BIP. In the present paper, we extend this methodology to the complete Coordination layer (Interactions and Priorities), which requires a more sophisticated notion of boolean constraints on *activation* and *firing* port variables, where firing variables are exactly the port variables we refer to in [8].

To the best of our knowledge, few authors in the domain of component-based design use activation or firing variables as we do in this paper, and we are not aware of any work making a combined use of both. The techniques closest to ours can be found in [20] and in the use of clocks in synchronous languages such as Lustre [21], Esterel [22], and Signal [23].

Several methodologies for synthesis of component coordination have been proposed in the literature, e.g., connector synthesis in [24,25,26]. In [24], connectors are synthesized in order to ensure deadlock freedom of systems that follow a given architectural style. The proposed methodology is seriously limited by two factors. Firstly, the proposed architectural style is very restrictive: components can only be connected in a specific way and communicate by passing two types of messages (notifications and requests). Secondly, in order to ensure deadlock freedom, the authors rely on the analysis that requires computing the product automaton representing the complete system, which is impractical for large systems.

In [25], Reo circuits are generated from constraint automata. This approach is limited, in the first place, by the complexity of building the automaton specification of interactions. An attempt to overcome this limitation is made in [26] by generating constraint automata from UML sequence diagrams. A commonly accepted problem of using UML, in general, and sequence diagrams, in particular, is the absence of formal semantics. Although the Reo approach effectively provides such semantics, there is no guarantee that this semantics can be understood by the designer, as the synthesized constraint automata can be rather complex. More importantly, synthesized Reo connectors (with the constraint automaton semantics) have state. Hence, they contribute to the state space explosion hampering verification of the final system properties.

Our approach allows one to directly synthesize stateless glue from safety requirements in a generic setting and circumvents the difficulty of behavior synthesis by applying the separation of concerns principle; it develops the line of [8] and has similarities with the synthesis of circuits from boolean specifications.

Finally, following the common practice, we use SOS rules to define composition (glue) operators. Since their introduction in [4], numerous formats for SOS rules have been proposed and extensively studied in order to provide generic properties of objects defined by SOS rules. The rule formats that we use in this paper represent two very simple special cases of the existing SOS formats. In particular, the format given by (1) is a special case of GSOS [3].

9 Conclusion

We proposed and studied a general framework for component-based construction of systems. The framework is based on an SOS-style characterization of composition operators, called glue operators. The presented boolean characterization of glue operators allows moving from heavy SOS-style definition to a lightweight boolean representation. The representation distinguishes between firing and activation variables. This distinction is essential for expressing priorities and, in general, situations where the state of one component influences enabledness of an interaction without participating in it. The use of the offer predicate ↑ is essential for taking into account offers of the composed individual behaviors. This leads to a concept of transition system richer than usual transition systems, which allows to distinguish between composite and atomic behavior.

The equivalence between glue constraints and universal glue can drastically simplify component-based design. We have shown through examples that glue constraints can be used to express given safety requirements. The synthesis of an implementation from such requirements can be automated by computing the firing-full DNF and the corresponding operational semantics rules. This provides means for deriving executable models from declarative safety requirements opening the way for an automated synthesis paradigm similar to hardware synthesis.

Contrary to the algebra presented in [2], the Algebra of Glue Constraints, is very general and intuitive. Indeed, the only axiom different from the standard Boolean ones is, basically, a sanity check: a port cannot participate if it is not active. Any formula in this algebra gives rise to a valid glue operator.

Furthermore, we have shown that, by focusing on the properties (expressed as glue constraints) that do not bear computation, we reduce a very hard and, in general, undecidable problem of synthesizing controllers to a tractable one. The Rescue Robot example shows that such properties are quite natural and can be expressed as glue constraints in a straightforward manner.

Amongst the formalisms discussed in the paper, BIP presents the advantage of being more appropriate for mastering the complexity of designs by developers. The principle of separating glue into two layers corresponding to interactions (glue with positive premises) and priorities (glue with negative premises and a single positive premise) drastically simplifies readability of specifications. The presented results suggest an evolution of BIP semantics to encompass the universal glue defined in this paper.

This work is part of a broader research project around BIP, including the development of the BIP language and associated tool-set for component-based design. The tool-set includes compilers as well as compositional verification tools. It has been successfully applied to modeling and validation of heterogeneous systems [11,27]. We will continue our study to further explore relations between three identified approaches: SOS-based, constraint-based, and layered.

References

1. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in BIP. In: 4^{th} IEEE Int. Conf. on Software Engineering and Formal Methods (SEFM 2006), pp. 3–12 (2006) Invited talk
2. Bliudze, S., Sifakis, J.: A notion of glue expressiveness for component-based systems. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 508–522. Springer, Heidelberg (2008)

3. Bloom, B.: Ready Simulation, Bisimulation, and the Semantics of CCS-Like Languages. PhD thesis, Massachusetts Institute of Technology (1989)
4. Plotkin, G.D.: A structural approach to operational semantics. Technical Report DAIMI FN-19, University of Aarhus (1981)
5. Mousavi, M., Reniers, M.A., Groote, J.F.: SOS formats and meta-theory: 20 years after. Theoretical Computer Science 373(3), 238–272 (2007)
6. Cheng, C.P., Fristoe, T., Lee, E.A.: Applied verification: The Ptolemy approach. Technical Report UCB/EECS-2008-41, University of California at Berkeley (2008)
7. Bliudze, S., Sifakis, J.: The algebra of connectors — Structuring interaction in BIP. In: Proc. of the EMSOFT 2007, pp. 11–20. ACM SigBED, New York (2007)
8. Bliudze, S., Sifakis, J.: Causal semantics for the algebra of connectors. Formal Methods in System Design 36(2), 167–194 (2010)
9. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: Annual IEEE Symposium on Foundations of Computer Science, vol. 2, pp. 746–757 (1990)
10. Lustig, Y., Vardi, M.Y.: Synthesis from component libraries. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 395–409. Springer, Heidelberg (2009)
11. Bensalem, S., Bozga, M., Nguyen, T.-H., Sifakis, J.: D-Finder: A tool for compositional deadlock detection and verification. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 614–619. Springer, Heidelberg (2009)
12. Balarin, F., et al.: Metropolis: An integrated electronic system design environment. IEEE Computer 36(4), 45–52 (2003)
13. Balasubramanian, K., et al.: Developing applications using model-driven design environments. IEEE Computer 39(2), 33–40 (2006)
14. Eker, J., et al.: Taming heterogeneity: The Ptolemy approach. Proc. of the IEEE 91(1), 127–144 (2003)
15. Bruni, R., Lanese, I., Montanari, U.: A basic algebra of stateless connectors. Theor. Comput. Sci. 366(1), 98–120 (2006)
16. Fribourg, L., Peixoto, M.V.: Concurrent constraint automata. In: ILPS 1993: Proc. of the 1993 International Symposium on Logic Programming, vol. 656. MIT Press, Cambridge (1993)
17. Baier, C., et al.: Modeling component connectors in Reo by constraint automata. Sci. Comput. Program. 61(2), 75–113 (2006)
18. Gößler, G., Sifakis, J.: Composition for component-based modeling. Sci. Comput. Program. 55(1–3), 161–183 (2005)
19. Montanari, U., Rossi, F.: Graph rewriting and constraint solving for modelling distributed systems with synchronization (extended abstract). In: Hankin, C., Ciancarini, P. (eds.) COORDINATION 1996. LNCS, vol. 1061, pp. 12–27. Springer, Heidelberg (1996)
20. Clarke, D., et al.: Deconstructing Reo. In: FOCLASA 2008. ENTCS (2008)
21. Halbwachs, N., et al.: The synchronous dataflow programming language LUSTRE. Proc. of the IEEE 79, 1305–1320 (1991)
22. Berry, G., Gonthier, G.: The ESTEREL synchronous programming language: Design, semantics implementation. Sci. Comput. Program. 19(2), 87–152 (1992)
23. Benveniste, A., Guernic, P.L., Jacquemot, C.: Synchronous programming with events and relations: the SIGNAL language and its semantics. Sci. Comput. Program. 16(2), 103–149 (1991)
24. Inverardi, P., Scriboni, S.: Connectors synthesis for deadlock-free component-based architectures. In: ASE 2001, pp. 174–181. IEEE Computer Society, Los Alamitos (2001)

25. Arbab, F., Baier, C., de Boer, F.S., Rutten, J., Sirjani, M.: Synthesis of Reo Circuits for Implementation of Component-Connector Automata Specifications. In: Jacquet, J.-M., Picco, G.P. (eds.) COORDINATION 2005. LNCS, vol. 3454, pp. 236–251. Springer, Heidelberg (2005)
26. Arbab, F., Meng, S.: Synthesis of connectors from scenario-based interaction specifications. In: Chaudron, M.R.V., Ren, X.-M., Reussner, R. (eds.) CBSE 2008. LNCS, vol. 5282, pp. 114–129. Springer, Heidelberg (2008)
27. Basu, A., et al.: Incremental component-based construction and verification of a robotic system. In: ECAI, pp. 631–635 (2008)