Composition for Component-Based Modeling

Gregor Gössler [a], Joseph Sifakis [b]

[a] *INRIA Rhône-Alpes, France*
[b] *VERIMAG, France*

Abstract

We propose a framework for component-based modeling using an abstract layered model for components. A component is the superposition of two models: a behavior model and an interaction model. Interaction models describe architectural constraints induced by connectors between components.

We propose and analyze general requirements for component composition that motivated and guided the development of the framework. We define an associative and commutative composition operator on components encompassing heterogeneous interaction. As a particular instance of the proposed framework, we consider components where behavior models are transition systems and interaction models are described by priority relations on interactions. This leads to a concept of "flexible" composition different from usual composition in that it preserves deadlock-freedom and is amenable to correctness by construction. Nevertheless, flexible composition is a partial operation. Product systems should be interaction safe in the sense that they do not violate constraints of the interaction model.

We propose results ensuring correctness by construction of a system from properties of its interaction model and of its components. The considered properties include global deadlock-freedom, individual deadlock-freedom of components, and interaction safety.

1 Introduction

Component-based engineering is essential for rigorous system design methodologies. It is founded on a paradigm which is common to all engineering disciplines: complex systems can be obtained by assembling components (building blocks). Components are usually characterized by abstractions that ignore

Email addresses: goessler@inrialpes.fr (Gregor Gössler), sifakis@imag.fr (Joseph Sifakis).

implementation details and describe properties relevant to their composition e.g. transfer functions, interfaces. Composition is used to build complex components from simpler ones. It can be formalized as an operation that takes in components and their integration constraints. From these, it provides the description of a new, more complex component.

Component-based engineering is widely used in VLSI circuit design methodologies, supported by a large number of tools. Software and system component-based techniques have known significant development, especially due to the use of object technologies supported by languages such as C++, Java, and standards such as UML and CORBA. However, these techniques have not yet achieved the same level of maturity as has been the case for hardware. For software components, it is not easy to establish a precise characterization of the service and functionality offered at their interface.

Existing software component technologies usually allow interaction by method calls under asynchronous execution. We lack semantic frameworks for component-based engineering encompassing meaningful integration of synchronous and asynchronous components, as well as use of various interaction mechanisms. This is the main obstacle to mastering the complexity of heterogeneous systems. It seriously limits the current state of the practice, as attested by the lack of system development platforms consistently integrating design activities, and the often prohibitive cost of validation.

The application of component-based design techniques raises two strongly related and hard problems.

First, the development of theory for building *complex heterogeneous systems*. Heterogeneity is in the different types of component interaction, such as strict (blocking) or non strict, data driven or event driven, atomic or non atomic and in the different execution models, such as synchronous or asynchronous.

Second, the development of theory for building systems which are *correct by construction*, especially with respect to essential generic properties such as deadlock-freedom and progress. In practical terms, this means that the theory supplies rules for reasoning on the structure of a system and for ensuring that such properties hold globally under some assumptions about its constituents e.g. components, connectors. Tractable correctness by construction results can provide significant guidance in the design process. Their lack leaves a posteriori verification of the designed system as the only means to ensure its correctness.

In this paper, we propose a framework for component-based modeling that brings some answers to the above issues. The framework uses an abstract layered model of components. It integrates and simplifies results about modeling timed systems by using timed automata with dynamic priorities [6,1].

A component is the superposition of two models: a behavior model and an interaction model.

- Behavior models describe the dynamic behavior of components.
- Interaction models describe architectural constraints on behavior. They are defined as a set of connectors and their properties. A connector is a maximal set of compatible component actions. The simultaneous occurrence of actions of a connector is an interaction.

An associative and commutative composition operator is defined on components. The operator builds two-layered components by composing separately the corresponding layers of its arguments. As a particular instance of the proposed framework, we consider components where behaviors are transition systems and interaction models are described by priority relations on interactions. This leads to a general framework for "flexible" composition which differs from existing ones such as process algebras [4,16,23,24], semantic frameworks for synchronous languages [5,14,3,22], and Statecharts [15].

The proposed composition distinguishes clearly between two different and orthogonal aspects of systems modeling: behavior and interaction (architecture). This distinction, apart from its methodological interest, allows solving technical problems such as associativity of a unique and powerful composition operator. The proposed framework has concepts in common with Metropolis [2] and Ptolemy [20] where a similar separation of concerns is advocated.

The proposed composition preserves deadlock-freedom. That is, if two components can perform some action from any state then their product does so. This is due to the fact that we replace restriction or other mechanisms often used to ensure strong synchronization between components, by dynamic priorities. Nevertheless, our composition is a partial operation: products must be *interaction safe*, that is, they do not violate strong synchronization assumptions. In that respect, our approach is has some similarity to [8].

The paper is organized as follows.

Section 2 discusses three requirements for composition in component-based modeling. The first is support for two main types of heterogeneity: heterogeneous interaction and heterogeneous execution. The second is that it provide results for ensuring correctness by construction for a few essential and generic system properties, such as deadlock-freedom. The third is the existence of a composition operator that allows abstraction and incremental description.

Section 3 presents composition and its properties.

Section 4 presents results ensuring correctness by construction of product systems from properties of their interaction model and of their components.

The considered properties include global deadlock-freedom (the product of deadlock-free components is deadlock-free), individual deadlock-freedom of components, and interaction safety.

Section 5 presents the application of correctness by construction results to an example.

Section 6 presents concluding remarks about the presented framework.

2 Requirements for Composition

2.1 General

We consider a very simple and abstract concept of components that is sufficient for the purpose of the study. A component can perform actions from a vocabulary of actions. A system of interacting components is a set of components integrated through various mechanisms for coordinating their execution. The overall effect of integration on the components of a system is the restriction of their behavior. It can be abstractly described by two types of integration constraints: interaction and execution constraints.

Interaction constraints characterize mechanisms used in architectures such as connectors, channels, synchronization primitives. Interactions result from the composition of actions.

Execution constraints restrict non determinism arising from concurrent execution, and ensure properties related to the efficiency of computation, such as synchronous execution and scheduling.

There exists a variety of formalisms proposing concepts for parallel execution of sequential entities, such as process algebras (CCS [24], CSP [16]), synchronous languages (Esterel, Lustre, Statecharts), hardware description languages (VHDL), system description languages (SystemC [25], Metropolis meta-model), and more general modeling languages (SDL [17], UML [13]). We use the term "component" to denote any executable description whose runs can be modeled as sequences of actions. Tasks, processes, threads, functions, blocks of code can be considered as components provided they meet these requirements.

The purpose of this section is to present concept requirements for composition in component-based modeling and to discuss the adequacy of existing formalisms with respect to these requirements.

There exist two main sources of heterogeneity: interaction and execution. Heterogeneity of interaction results from the combination of different kinds of interaction.

Interactions can be *atomic* or *non atomic*. The effect of atomic interactions on participating components cannot be altered through interference with other interactions. Process algebras and synchronous languages assume atomic interactions. In languages with buffered communication (SDL, UML) or in multithreaded languages (Java), interactions are not atomic, in general.

Interactions can involve *strict* or *non strict* synchronization. For instance, CSP rendez-vous are strict interactions; their execution requires participation of the involved actions. Strict synchronization can introduce deadlocks in systems of interacting deadlock-free components. If a component persistently offers an action and its environment is unable to offer matching actions, then there is a risk of deadlock. In synchronous languages, interactions are atomic and non strict as execution of outputs does not require synchronization with inputs. Nevertheless, for some input to be triggered, a matching output is necessary.

Heterogeneity of execution results from the combination of two execution paradigms.

Synchronous execution is typically adopted in hardware, in synchronous languages, and in time triggered architectures and protocols. It considers that a system execution is a sequence of steps. It assumes synchrony, meaning that the system's environment does not change during a step, or equivalently "that the system is infinitely faster than its environment". In each execution step, all the system components contribute by executing some "quantum" computation. The synchronous execution paradigm has a built-in strong assumption of fairness: in each step all components execute a quantum computation defined by using either quantitative or logical time.

The *asynchronous* paradigm does not adopt any notion of global computation step in a system's execution. It is used in languages for the description of distributed systems such as SDL and UML, and programming languages such as Ada and Java. The lack of built-in mechanisms for sharing computation between components can be compensated through scheduling. This paradigm is also common to all execution platforms supporting multiple threads, tasks, etc.

Currently, there is no unified framework encompassing heterogeneous composition. Figure 1 shows existing languages in a three-dimensional space with coordinates corresponding to execution (synchronous/asynchronous) and to inter-

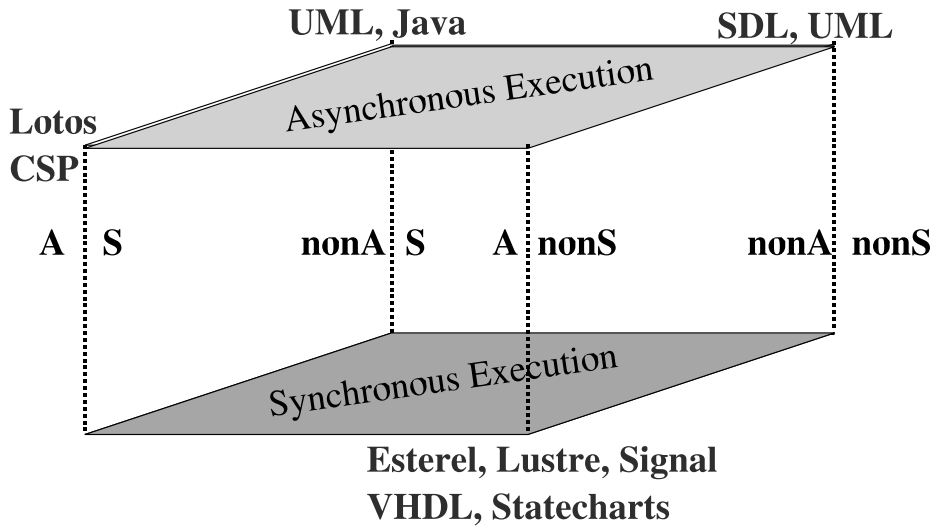

Fig. 1. About composition: heterogeneity. A: atomic, S: strict interaction.

action: atomic/non atomic and strict/non-strict. It is worth noting that synchronous languages use non strict and atomic interactions. This choice seems appropriate for synchronous execution. On the contrary, for asynchronous execution there is no language using this kind of interaction.

2.3 Correctness by Construction

Frameworks for component-based modeling should provide methods for establishing correctness by construction, in particular for classes of very common and generic properties such as deadlock-freedom and liveness. In principle, two types of rules are needed for establishing correctness by construction.

- *Composability* rules allowing to infer that, under some local conditions, a component property will remain valid after composition. These rules are essential for preserving previously established component properties. For instance, to guarantee that a component without internal deadlocks will remain deadlock-free after composition. Composability is essential for incremental system construction. It means stability of component properties across integration (when its environment changes by adding or removing components). Property instability phenomena are currently poorly understood e.g. feature interaction in telecommunications, or non composability of scheduling algorithms. Results in composability are badly needed.
- *Compositionality* rules allowing to infer overall system properties from its components' properties. Existing compositionality results deal mainly with preservation of safety properties [19,10,9]. Compositionality results for progress properties are essential for the correctness of reactive systems.

It is often necessary to modify components according to the context of their use, at the risk of altering their behavior. Such modifications may be needed to adapt them to a particular type of composition. For instance, if a composition operator allows only strict interaction, this operator can be used for non strict interaction by modifying both the interface and the behavior of the components in the following manner (see for instance Milner's SCCS [23]): For each action a in the interface add a "complementary" action $\bar{a}$ that will be executed from all the states from which a is not possible. Conversely, modeling strict interactions by using non strict interactions requires similar modifications of the components.

We currently lack sufficiently powerful and abstract composition operators encompassing all kinds of interaction without modification of the integrated components.

Another important requirement for composition is incrementality of description. Incrementality means that models can be constructed by adding or removing components and that the result of the construction is independent of the order of integration. Associative and commutative composition operators allow incrementality.

Existing theoretical frameworks for composition such as CCS and SCCS, use parallel composition operators that are associative and commutative. Nevertheless, these operators are not expressive enough and need to be combined with other operators such as hiding and restriction. This jeopardizes incrementality of description. For instance, if restriction is used in a system's model, its integration in a larger model may need changing the scope of restriction.

Lack of incrementality is also a well-identified problem in graphical formalisms such as Statecharts or UML. Their operational semantics associate with descriptions global transition systems by using implicitly n-ary composition operators (n is equal to the number of the composed components.

The definition of an associative and commutative composition operator which is expressive and abstract enough to support heterogeneous integration remains an open problem.

3 Composition

We present a modeling framework based on a binary associative and commutative composition operator for heterogeneous interaction. For the sake of simplicity, atomic components are characterized by a set of actions and the associated behavior.

3.1 Interaction Models

Composition operators allow to build a system as a set of components that interact by respecting constraints of an *interaction model*. The latter characterizes a system architecture as a set of connectors and their properties. Roughly speaking, connectors relate actions of different components and can be abstractly represented as maximal sets of interacting actions (interactions).

Consider a set of components with disjoint vocabularies of actions A_i for $i \in K$, K a set of indices. We put $A = \bigcup_{i \in K} A_i$.

A *connector* c is a non empty subset of A such that $\forall i \in K \; . \; |A_i \cap c| \leqslant 1$. A connector defines a maximally compatible set of interacting actions. For the sake of generality, our definition accepts singleton connectors. The use of the connector $\{a\}$ in a description is interpreted as the fact that action a cannot be involved in interactions with other actions (is an internal action).

Given a connector c, an *interaction* α of c is any term of the form $\alpha = a_1 \mid \ldots \mid a_n$ such that $\{a_1, \ldots, a_n\} \subseteq c$. As usual [23,4], we assume that $\mid$ is a binary associative and commutative operator used to denote some abstract and partial action composition operation. The interaction $a_1 \mid \ldots \mid a_n$ is the result of the occurrence of the actions $a_1, \ldots, a_n$. When α and α' are interactions we write $\alpha \mid \alpha'$ to denote the interaction resulting from their composition (if its is defined).

Notice that if $\alpha = a_1 \mid \ldots \mid a_n$ is an interaction then any term corresponding to a sub-set of $\{a_1, \ldots, a_n\}$ is an interaction. By analogy, we say that α' is a sub-interaction of α if $\alpha = \alpha' \mid \alpha''$ for some interaction α''. Clearly, actions are minimal interactions.

The set of the interactions of a connector $c = \{a_1, \ldots, a_n\}$, denoted by $I(c)$, consists of all the interactions corresponding to sub-sets of c (all the sub-interactions of c). We extend the notation to sets of connectors. If C is a set of connectors then $I(C)$ is the set of its interactions. Clearly for C_1, C_2 sets of connectors, $I(C_1 \cup C_2) = I(C_1) \cup I(C_2)$.

Definition 1 (Set of connectors) *The* set of connectors *of a system con-*

sisting of a set of components K with disjoint action vocabularies A_i for $i \in K$, is a set C such that $\bigcup_{c \in C} c = \bigcup_{i \in K} A_i$, and if $c \in C$ then there exists no $c' \in C$ and $c \subsetneq c'$. That is, C contains only maximal sets.

Definition 2 (Interaction model) *The* interaction model *of a system consisting of a set of components K with a set of connectors C is a pair $IM = (I(C), I(C)^+)$ where $I(C)^+ \subseteq I(C)$, is the set of the* complete *interactions such that $\forall b, b' \in I(C)$, $b \in I(C)^+$ and $b \subseteq b'$ implies $b' \in I(C)^+$. We denote by $I(C)^-$ the set of the incomplete (non complete) interactions.*

Notice that any action appears in some connector. The requirement that C contains only maximal sets ensures a bijection between the set of connectors C and the corresponding set of interactions $I(C)$. Given $I(C)$, the corresponding set of connectors is uniquely defined and is C. To simplify notation, we write IC instead of $I(C)$.

The distinction between complete and incomplete interactions introduces a notion of correctness which is essential for systems of interacting components. As models are built incrementally, interactions are obtained by successively composing actions. It is often necessary for a given system, to express the fact that some interaction of a sub-system is not a legal interaction of the overall system. This is typically the case for binary strict synchronization (rendez-vous) between two actions *send* and *receive* offered by two components. These actions should be considered as incomplete and *send*|*receive* as complete. If a system model involving the two components can execute actions *send* or *receive* then it is not correct as it violates the rendez-vous assumption about strict synchronization.

The execution of a complete interaction by a component does not require synchronization with interactions of its environment (other components). The execution of an incomplete interaction requires synchronization with some other interaction to produce a larger one which may be either complete or incomplete. Thus, incompleteness implies the obligation to synchronize with matching interactions as specified by the connectors. If an interaction model has no complete interactions, then components must synchronize to produce (incomplete) interactions which are maximal in the sense that they involve all the actions in some connector. We consider that in a system, only complete or maximal incomplete interactions are legal. This induces a notion of correctness called *interaction safety*, defined in 4.2.

In our framework the distinction between complete and incomplete interactions is used to encompass distinctions such as output/input, internal/external, uncontrollable/controllable used in different modeling formalisms. Clearly, internal actions of components should be considered as complete because they can be performed independently of the state of their environment. In some for-

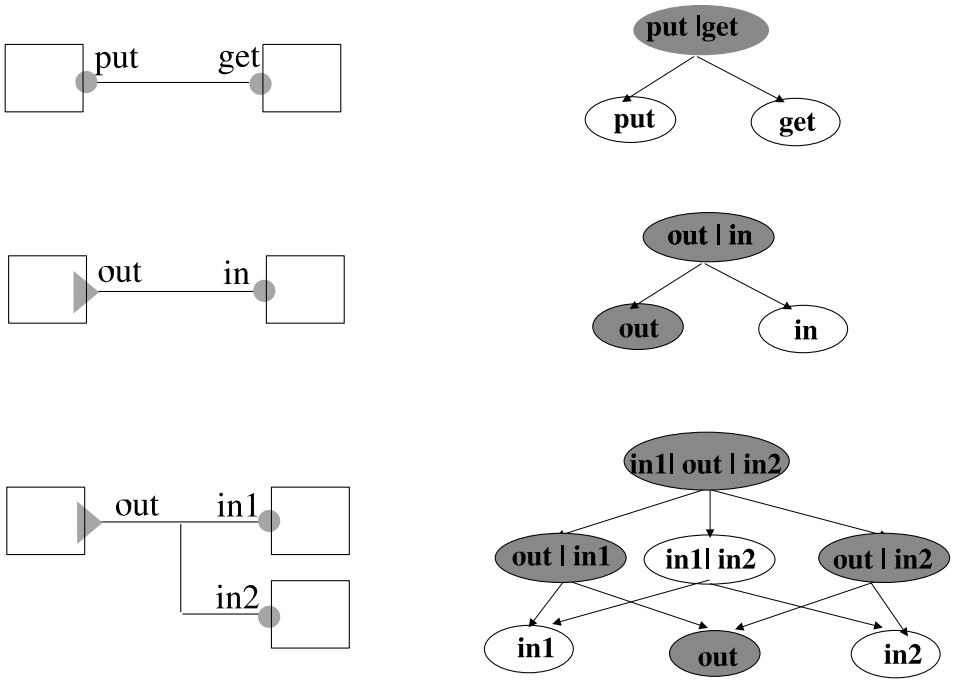

Fig. 2. Flexible composition: interaction structure.

malisms, output actions are complete (synchronous languages, asynchronous buffered communication). In some others such as CSP and Lotos, all synchronizing actions are incomplete.

A property about complete interactions is closedness for containment that is, if α is a complete interaction then any interaction containing it, is complete. This property is motivated by both pragmatic and technical considerations. It ensures consistency of the composition of interaction models and has a "natural" interpretation. If α is a complete interaction of a component, then the fact that $\alpha \cup \alpha'$ remains complete is consistent with the assumption that the component can execute α independently of the state of its environment.

Very often it is sufficient to consider that the interactions of IC^+ are defined from a given set of complete actions $A^+ \subseteq A$. That is, IC^+ consists of all the interactions of IC where at least one complete action (element of A^+) is involved. In the example of figure 2, we give sets of connectors and complete actions to define interaction models. By convention, bullets represent incomplete actions and triangles complete actions. In the partially ordered set of the interactions, full nodes denote complete interactions. The interaction between *put* and *get* represented by the interaction $put|get$ is a rendez-vous meaning that synchronization is blocking for both actions. The interaction between *out* and *in* is asymmetric as *out* can occur alone even if *in* is not possible. Nevertheless, the occurrence of *in* requires the occurrence of *out*. The interactions between *out*, in_1 and in_2 are asymmetric. The output *out* can occur alone or in synchronization with any of the inputs in_1, in_2.

In general, completeness of interactions need not be the consequence of the completeness of some action. For instance, consider a connector $\{a_1, a_2, a_3, a_4\}$ and suppose that the set of the minimal complete interactions of $I\{a_1, a_2, a_3, a_4\}$ is $a_1|a_2$ and $a_3|a_4$. That is, the actions a_1, a_2, a_3, a_4 are incomplete and only interactions containing $a_1|a_2$ or $a_3|a_4$ are complete. This specification requires strict synchronization of at least one of the two pairs (a_1, a_2), (a_3, a_4).

3.2 Composition of Interaction Models

Consider the interaction model $IM = (IC, IC^+)$ of a set of interacting components K with disjoint action vocabularies A_i for $i \in K$. IC and IC^+ denote the sets of interactions and complete interactions, respectively on the vocabulary of actions $A = \bigcup_{i \in K} A_i$.

Definition 3 (Set of connectors of a partition) *The set of the connectors of a partition $K_1, \ldots, K_n$ of K, is a set $C[K_1, \ldots, K_n]$ of connectors having at least one action in each set of components, that is, $C[K_1, \ldots, K_n] = \{c = c_1 \cup \cdots \cup c_n \mid \forall i \in [1, n] \,.\, c_i \in C[K_i] \ \wedge \ c \in C[K]\}$.*

Clearly, $C[K_1, \ldots, K_n]$ is the set of the connectors of $IM[K_1 \cup \cdots \cup K_n]$ which are not connectors of any $IM[K']$ for any subset K' of at most $n-1$ elements from $\{K_1, \ldots, K_n\}$.

Definition 4 (Interaction model of a partition) *The interaction model of a partition $K_1, \ldots, K_n$ of $K' \subseteq K$, with disjoint action vocabularies A_i for $i \in K'$ and set of connectors $C[K_1, \ldots, K_n]$, is a pair $IM[K_1, \ldots, K_n] = (IC[K_1, \ldots, K_n], IC[K_1, \ldots, K_n]^+)$ where $IC[K_1, \ldots, K_n]^+ = IC[K_1, \ldots K_n] \cap IC^+$.*

Notice that when the partition consists of only one set, then the above definitions agree with Definitions 1 and 2.

Remark 1 *Writing the interaction model of a sub-system $K' \subseteq K$ as the projection of IM on K' helps to simplify notation. It does not preclude incremental construction. The following proposition provides a basis for computing the interaction model $IM[K_1 \cup K_2]$ from the interaction models $IM[K_1]$ and $IM[K_2]$ and from the interaction model of the connectors relating components of K_1 and K_2.*

Proposition 1 *Given K_1, K_2, a partition of K, interaction models $IM[K_i] = (IC[K_i], IC[K_i]^+)$, for $i = 1, 2$, the interaction model $IM[K_1 \cup K_2] = (IC[K_1 \cup$*

$K_2], IC[K_1 \cup K_2]^+)$ *can be defined by*

$$\begin{aligned} IC[K_1 \cup K_2] &= IC[K_1] \cup IC[K_2] \cup IC[K_1, K_2] \\ IC[K_1 \cup K_2]^+ &= IC[K_1]^+ \cup IC[K_2]^+ \cup IC[K_1, K_2]^+ \\ IM[K_1 \cup K_2] &= (IC[K_1 \cup K_2], IC[K_1 \cup K_2]^+) \\ &= IM[K_1] \cup IM[K_2] \cup IM[K_1, K_2] \end{aligned}$$

where $IC[K_1, K_2]^+$ *is any set of interactions of the form* $IC[K_1, K_2]^+ = IC[K_1]^+ \cap IC[K_1, K_2] \cup IC[K_2]^+ \cap IC[K_1, K_2] \cup I$ *with* $I \subseteq IC[K_1, K_2] \setminus (IC[K_1]^+ \cup IC[K_2]^+)$ *and* $\alpha \in IC[K_1]^+ \cup IC[K_2]^+$, $\alpha_2 = \alpha \imath \alpha_1 \in IC[K_1, K_2] \setminus (IC[K_1]^+ \cup IC[K_2]^+)$ *implies* $\alpha_2 \in I$.

Proof. The first equality comes from the fact that $C[K_1] \cup C[K_2] \cup C[K_1, K_2]$ contains all the connectors of $C[K_1 \cup K_2]$ and other sets of interactions that are not maximal. By definition, IC contains all the sub-sets of C. Thus, $IC[K_1 \cup K_2] = I(C[K_1] \cup C[K_2] \cup C[K_1, K_2]) = IC[K_1] \cup IC[K_2] \cup IC[K_1, K_2]$.

To prove the second equality it is sufficient to prove that $IC[K_1 \cup K_2]^+ \subseteq IC[K_1 \cup K_2]$ and $IC[K']^+ = IC[K_1 \cup K_2]^+ \cap IC[K']$ for $[K'] = [K_1], [K_2], [K_1, K_2]$. This is easy to check given that $IC[K_1]^+ \cup IC[K_2]^+ \cup IC[K_1, K_2]^+ = IC[K_1]^+ \cup IC[K_2]^+ \cup I$. ■

Property 1 *Given* K_1, K_2, K_3 *three disjoint subsets of a set of components* K, *and the interaction models* $IM[K'] = (IC[K'], IC[K']^+)$, *for* $[K'] = [K_1, K_3], [K_2, K_3], [K, _1, K_2, K_3]$,

$$\begin{aligned} IC[K_1 \cup K_2, K_3] &= IC[K_1, K_3] \cup IC[K_2, K_3] \cup IC[K_1, K_2, K_3] \\ IM[K_1 \cup K_2, K_3] &= IM[K_1, K_3] \cup IM[K_2, K_3] \cup IM[K_1, K_2, K_3] \end{aligned}$$

Proof. The first equality comes from the fact that $C[K_1, K_3] \cup C[K_2, K_3] \cup C[K_1, K_2, K_3]$ contains all the connectors of $C[K_1 \cup K_2, K_3]$ and in addition, other sets of interactions that are not maximal. By definition, IC contains all the sub-sets of C. Thus, $IC[K_1 \cup K_2, K_3] = I(C[K_1, K_3] \cup C[K_2, K_3] \cup C[K_1, K_2, K_3])$ from which we get the result by distributivity of I over union.

The second equality results from the fact that $IC[K_1 \cup K_2, K_3]^+ = IC[K_1, K_3]^+ \cup IC[K_2, K_3]^+ \cup IC[K_1, K_2, K_3]^+ = IC[K_1 \cup K_2 \cup K_3]^+ \cap IC[K_1 \cup K_2, K_3]$. ■

This property allows computing the connectors and thus the interactions between $IM[K_1 \cup K_2]$ and $IM[K_3]$ in terms of the interactions between $IM[K_1]$, $IM[K_2]$, and $IM[K_3]$. Is is used to obtain the following expansion formula:

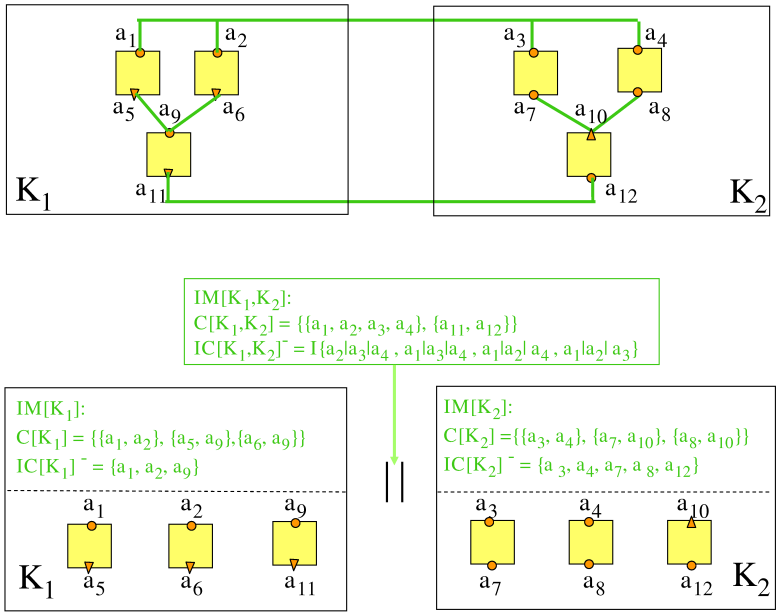

Fig. 3. The composition principle.

Proposition 2 (Expansion formula)

$$IM[K_1 \cup K_2 \cup K_3] = IM[K_1] \cup IM[K_2] \cup IM[K_3] \cup IM[K_1, K_2] \cup IM[K_1, K_3] \cup IM[K_2, K_3] \cup IM[K_1, K_2, K_3] \ .$$

3.3 Composition Semantics and Properties

We consider that a system S is a pair $S = (B, IM)$ where B is the behavior model of S and IM is its interaction model. As in the previous section, IM is the interaction model of a set of interacting components K with disjoint action vocabularies A_i, $i \in K$.

For given $K' \subseteq K$, we denote by $S[K']$ the sub-system of S consisting of components of K', $S[K'] = (B[K'], IM[K'])$, where $IM[K']$ is defined as before.

We define an abstract composition operator $\|$ allowing to obtain for disjoint sub-sets K_1, K_2 of K, the system $S[K_1 \cup K_2]$ as the composition of the sub-systems $S[K_1]$, $S[K_2]$ for given interaction model $IM[K_1, K_2]$ connecting the two sub-systems. The operator composes separately the behavior models and the interaction models of the sub-systems.

Definition 5 *The composition of two systems $S[K_1]$ and $S[K_2]$ is the system*

$S[K_1 \cup K_2] = (B[K_1], IM[K_1]) \| (B[K_2], IM[K_2]) = (B[K_1] \times B[K_2], IM[K_1] \cup IM[K_2] \cup IM[K_1, K_2])$ *where* $\times$ *is a binary associative behavior composition operator such that* $B[K_1] \times B[K_2] = B[K_1 \cup K_2]$.

Remark 2 *This definition does not make any specific assumption about behavior models which can be programs, state equations, formulas of a temporal logic or any description representing transition relations or their abstractions. Clearly, the choice of a behavior model implies an adequate interpretation of the interaction model. If for instance, components are circuits, their behavior can be described by boolean state equations and interactions are correspondences between inputs and outputs. Composition of behaviors is the (disjoint) union of the state equations of the components.*

Due to proposition 1 we have $(B[K_1], IM[K_1]) \| (B[K_2], IM[K_2]) = (B[K_1 \cup K_2], IM[K_1 \cup K_2])$, which means that composition of sub-systems gives the system corresponding to the union of their components.

Notice that under these assumptions composition is associative:

$$\begin{aligned}
&\Big((B[K_1], IM[K_1]) \| (B[K_2], IM[K_2])\Big) \| (B[K_3], IM[K_3]) = \\
&= (B[K_1 \cup K_2], IM[K_1 \cup K_2]) \| (B[K_3], IM[K_3]) \\
&= (B[K_1] \times B[K_2] \times B[K_3], IM[K_1 \cup K_2] \cup IM[K_3] \cup IM[K_1 \cup K_2, K_3]) \\
&= (B[K_1 \cup K_2 \cup K_3], IM[K_1 \cup K_2 \cup K_3])
\end{aligned}$$

by application of proposition 2.

3.3.1 Transition Systems with Priorities

We consider the particular case where interactions are atomic, component behaviors are transition systems, and the constraints are modeled as priority orders on interactions. Transition systems with priorities have already been studied and used to model timed systems. The interested reader can refer to [7,1].

Definition 6 (Transition system) *A* transition system *B is a triple $(Q, I(A), \rightarrow)$ where Q is a set of states, $I(A)$ is a set of interactions on the action vocabulary A, and $\rightarrow \subseteq Q \times I(A) \times Q$ is a transition relation.*

As usual, we write $q_1 \xrightarrow{\alpha} q_2$ instead of $(q_1, \alpha, q_2) \in \rightarrow$.

Definition 7 (Transition system with priorities) *A* transition system with priorities *is a pair $(B, \prec)$ where B is a transition system with set of interactions $I(A)$, and $\prec$ is a* priority order*, that is, a strict partial order on $I(A)$.*

Semantics: A transition system with priorities represents a transition system: if $B = (Q, I(A), \rightarrow)$, then $(B, \prec)$ represents the transition system $B' = (Q, I(A), \rightarrow')$ such that $q_1 \stackrel{\alpha}{\rightarrow}' q_2$ if $q_1 \stackrel{\alpha}{\rightarrow} q_2$ and there exist no α' and q_3 such that $\alpha \prec \alpha'$ and $q_1 \stackrel{\alpha'}{\rightarrow} q_3$.

Definition 8 ($\oplus$) *The sum $\prec^1 \oplus \prec^2$ of two priority orders $\prec^1$, $\prec^2$ is the least priority order (if it exists) such that $\prec^1 \cup \prec^2 \subseteq \prec^1 \oplus \prec^2$.*

Notice that $\prec^1 \oplus \prec^2$, if it is defined, is the transitive closure of $\prec^1 \cup \prec^2$.

Lemma 1 *$\oplus$ is a (partial) associative and commutative operator.*

Definition 9 ($\|$) *Consider a system $S[K]$ with interaction model $IM[K] = (IC[K], IC[K]^+)$. Let K_1, K_2 be two disjoint subsets of K and $S[K_1] = (B[K_1], \prec^1)$, $S[K_2] = (B[K_2], \prec^2)$ be two sub-systems of $S[K]$ such that their priority orders do not allow domination of complete interactions by incomplete ones, that is for all $\alpha \in IC[K]^+$ and $\alpha' \in IC[K]^-$, $\neg(\alpha \prec^i \alpha')$ for $i = 1, 2$.*

If $B_i = (Q_i, IC[K_i], \rightarrow_i)$ for $i = 1, 2$, then $S[K_1] \| S[K_2]$ is the composition of $S[K_1]$ and $S[K_2]$ defined by $S[K_1] \| S[K_2] = (B_1 \times B_2, \prec^1 \oplus \prec^2 \oplus \prec^{12})$, where

$B_1 \times B_2 = (Q_1 \times Q_2, IC[K_1 \cup K_2], \rightarrow_{12})$ with

$$q_1 \stackrel{\alpha}{\rightarrow}_1 q_1' \text{ implies } (q_1, q_2) \stackrel{\alpha}{\rightarrow}_{12} (q_1', q_2)$$
$$q_2 \stackrel{\alpha}{\rightarrow}_2 q_2' \text{ implies } (q_1, q_2) \stackrel{\alpha}{\rightarrow}_{12} (q_1, q_2')$$
$$q_1 \stackrel{\alpha_1}{\rightarrow}_1 q_1' \text{ and } q_2 \stackrel{\alpha_2}{\rightarrow}_2 q_2' \text{ implies } (q_1, q_2) \stackrel{\alpha_1 | \alpha_2}{\rightarrow}_{12} (q_1', q_2') \text{ if } \alpha_1 | \alpha_2 \in IC[K_1 \cup K_2].$$

$\prec^{12}$ is the minimal priority order on $IC[K_1 \cup K_2]$ such that

- *$\alpha_1 \prec^{12} \alpha_1 | \alpha_2$ for $\alpha_1 | \alpha_2 \in IC[K_1, K_2]$ (maximal progress priority rule);*
- *$\alpha_1 \prec^{12} \alpha_2$ for $\alpha_1 \in IC[K_1 \cup K_2]^{--}$ and $\alpha_2 \in IC[K_1 \cup K_2]^+$, where $IC[K_1 \cup K_2]^{--}$ denotes the elements of $IC[K_1 \cup K_2]^-$ that are non-maximal in $IC[K_1 \cup K_2]$ (completeness priority rule).*

The first priority rule favors the largest interaction. The second allows ensuring correctness of the model. Non maximal incomplete (illegal) interactions are prevented if complete interactions are possible in the product: if a component can perform a complete interaction then all non maximal incomplete interactions are blocked.

Notice that priority rules of $\prec^{12}$ allow preventing illegal actions of the product which are usually eliminated by using restriction operators [23,24]. A main difference between the two approaches is that restrictions remove all illegal interactions from the product while priorities eliminate illegal interactions only if they are dominated by complete interactions in the priority order. The

use of restriction may introduce deadlocks in the product of deadlock-free components while deadlock-freedom is preserved by using priorities as shown in the next section.

Proposition 3 $\|$ *is a total, commutative and associative operator.*

Proof. Total operator: prove that for $K_1 \cap K_2 = \emptyset$, $\prec^1 \oplus \prec^2 \oplus \prec^{12}$ is a priority order, that is, the transitive closure of the union of $\prec^1$, $\prec^2$, and $\prec^{12}$ does not have any circuits.

The maximal progress priority rule defines a priority order isomorphic to the set inclusion partial order, and is thus circuit-free.

The completeness priority rule relates incomplete and complete interactions and is circuit-free, too. The only source of a priority circuit could be the existence of interactions $\alpha_1, \alpha_2, \alpha_3 \in IC[K_1 \cup K_2]$ such that $\alpha_1 = \alpha_2 | \alpha_3$, $\alpha_1 \in IC[K_1 \cup K_2]^{--}$, and $\alpha_2 \in IC[K_1 \cup K_2]^{+}$. This is impossible due to the monotonicity requirement of definition 2.

Associativity:

$$\begin{aligned}
&\Big((B[K_1],\ \prec^1) \parallel (B[K_2],\ \prec^2)\Big) \parallel (B[K_3],\ \prec^3) = \\
&= (B[K_1 \cup K_2],\ \prec^1 \oplus \prec^2 \oplus \prec^{12}) \parallel (B[K_3],\ \prec^3) \\
&= (B[K_1 \cup K_2 \cup K_3],\ \prec^1 \oplus \prec^2 \oplus \prec^{12} \oplus \prec^3 \oplus \prec^{[12],3})
\end{aligned}$$

where $\prec^{12}$ is the least priority order defined by

- $\alpha_1 \prec^{12} \alpha_1 | \alpha_2$ for $\alpha_1 | \alpha_2 \in IC[K_1, K_2]$, and
- $\alpha_1 \prec^{12} \alpha_2$ for $\alpha_1 \in IC[K_1 \cup K_2]^{--}$ and $\alpha_2 \in IC[K_1 \cup K_2]^{+}$,

and $\prec^{[12],3}$ is the least priority order defined by

- $\alpha_1 \prec^{[12],3} \alpha_1 | \alpha_2$ for $\alpha_1 | \alpha_2 \in IC[K_1 \cup K_2, K_3]$, and
- $\alpha_1 \prec^{[12],3} \alpha_2$ for $\alpha_1 \in IC[K_1 \cup K_2 \cup K_3]^{--}$ and $\alpha_2 \in IC[K_1 \cup K_2 \cup K_3]^{+}$.

Let $\prec$ be the order defined by

- $\alpha_1 \prec \alpha_1 | \alpha_2$ for $\alpha_1 | \alpha_2 \in IC[K_1, K_2] \cup IC[K_1, K_3] \cup IC[K_2, K_3] \cup IC[K_1, K_2, K_3]$, and
- $\alpha_1 \prec \alpha_2$ for $\alpha_1 \in IC[K_1 \cup K_2 \cup K_3]^{--}$ and $\alpha_2 \in IC[K_1 \cup K_2 \cup K_3]^{+}$.

By comparing both relations it is clear that $\prec^{12} \cup \prec^{[12],3} = \prec$ (by using the fact that $IC[K_1, K_2, K_3] \subseteq IC[K_1 \cup K_2, K_3]$).

We show that if $\prec^+$ is the transitive closure of $\prec$ then $\prec^+ = \prec$. Consider interactions α_1 and α_2 such that $\alpha_1 | \alpha_2 \in IC[K_1, K_2] \cup IC[K_1, K_3] \cup IC[K_2, K_3] \cup$

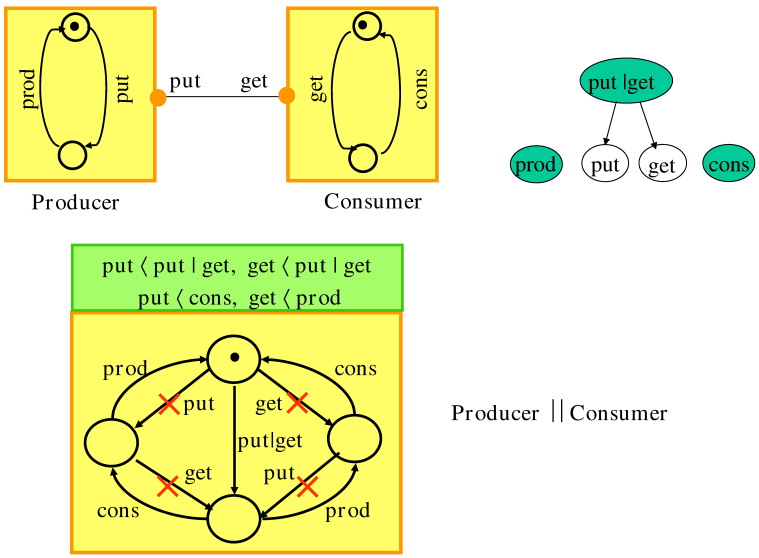

Fig. 4. Composition: producer/consumer.

$IC[K_1, K_2, K_3]$. If $\alpha_1 \in IC[K_1 \cup K_2 \cup K_3]^+$ then by definition 2, $\alpha_1 | \alpha_2 \notin IC[K_1 \cup K_2 \cup K_3]^{--}$. Conversely, if $\alpha_1 | \alpha_2 \in IC[K_1 \cup K_2 \cup K_3]^{--}$ then $\alpha_1 \notin IC[K_1 \cup K_2 \cup K_3]^+$. Thus, $\prec^+ = \prec$, and $\prec^{12} \oplus \prec^{[12],3} = \prec$. So the resulting priority order $\prec^1 \oplus \prec^2 \oplus \prec^3 \oplus \prec$ is the same independently of the order of composition. ■

Example 1 *Consider the system consisting of a producer and a consumer. The components interact by rendez-vous. The actions* put *and* get *are incomplete. We assume that the actions* prod *and* cons *are internal and thus complete. Figure 4 gives the interaction model corresponding to these assumptions. The product system consists of the product transition system and the priority order defined from the interaction model. The priority order removes all incomplete actions (crossed transitions).*

4 Correctness by Construction

4.1 Global Deadlock-Freedom

We give basic results about deadlock-freedom preservation for transitions systems with priorities. Similar results have been obtained for timed transition systems with priorities in [6].

Definition 10 (Deadlock-freedom) *A transition system* $B = (Q, I(A),$

$\rightarrow$) *is called* deadlock-free *if it has no sink states, that is if for any state* q *there exist* α, q' *such that* $(q, \alpha, q') \in \rightarrow$. *A system is deadlock-free if the transition system with priorities representing it is deadlock-free.*

Proposition 4 (Composability) *Deadlock-freedom is preserved by priority orders that is, if* B *is deadlock-free then* $(B, \prec)$ *is deadlock-free for any priority order* $\prec$.

Proposition 5 (Compositionality) *Deadlock-freedom is preserved by composition that is, if* $(B_1, \prec^1)$ *and* $(B_2, \prec^2)$ *are deadlock-free then* $(B_1, \prec^1) \| (B_2, \prec^2)$ *is deadlock-free, if* $\prec^1$, $\prec^2$ *meet the requirements of Definition 9.*

Proof. Follows from the fact that composition of behaviors preserves deadlock-freedom and from the previous proposition. ■

Corollary 1 *Any system obtained by composition of deadlock-free components is deadlock-free.*

4.2 Interaction Safety

As explained in section 3.1, the distinction between complete and incomplete interactions is essential for building correct models. Our composition operation preserves deadlock-freedom of components but it does not prevent the occurrence of non maximal incomplete interactions.

We introduce a notion of correctness called *interaction safety*.

Definition 11 (Interaction safety) *Consider a system* S *with interaction model* $IM = (IC, IC^+)$. *Define the priority order* $\prec$ *on incomplete interactions such that* $\alpha_1 \prec \alpha_2$ *if* $\alpha_1 \in IC^{+-}$ *and* $\alpha_2 \in IC^{+} \setminus IC^{+-}$. S *is called* interaction safe *if its restriction by* $\prec$ *can perform only complete or maximal incomplete interactions.*

Notice that the rule defining the priority order $\prec$ is similar to the completeness priority rule of definition 9. For a given system, incomplete interactions that are maximal in IC have the same status as complete interactions with respect to non maximal incomplete interactions. Nevertheless, maximality of incomplete interactions depends on the overall interaction model. For instance, consider a system consisting of three components with a connector $\{a_1, a_2, a_3\}$ such that all its interactions are incomplete. The interaction $a_1 | a_2$ is legal in the sub-system consisting of the first two components while it is illegal in the overall system. In the latter, $a_1 | a_2$ is incomplete and non maximal. It must synchronize with a_3 to produce the maximal incomplete interaction $a_1 | a_2 | a_3$.

We give below a method for checking whether a model is interaction safe. Notice that the same method can be used to check deadlock-freedom of systems of deadlock-free interacting components, for composition operators using restriction as in [24]. If a system is not interaction safe then there exists a state from which it can perform only illegal interactions. This is a deadlock state for composition with restriction.

Dependency graph: Consider a system $S[K]$ consisting of a set of interacting components K with interaction model $IM = (IC, IC^+)$. For $c \in C$ (C is the set of the connectors of IC) we denote by $I^+_{\min}(c)$ the set of the minimal complete interactions of c, and write $I^+_{\min}(C)$ for $\{i \in I^+_{\min}(c)\}_{c \in C}$.

The dependency graph of $S[K]$ is a labeled bipartite graph with two sets of nodes: the components of K, and nodes labeled with elements of the set $\{(c, \alpha(c)) \mid c \in C \ \wedge \ I^+_{\min}(c) = \emptyset\} \cup \{(c, \alpha) \mid c \in C \ \wedge \ \alpha \in I^+_{\min}(c)\}$, where $\alpha(c)$ is the maximal interaction of c (involving all the elements of c).

The edges are labeled with actions of A as follows:

Let $(c, \alpha) = (\{a_1, \ldots, a_n\}, \alpha)$ be a node of the graph and denote by $owner(a_i)$ the component which is owner of action a_i. For all actions a_i of c occurring in α, add an edge labeled with a_i from $owner(a_i)$ to (c, α). For all actions a_i of c, add an edge labeled with a_i from (c, α) to $owner(a_i)$ if a_i is offered in some *incomplete state* of $owner(a_i)$, that is, a state in which no complete or maximal interaction is offered.

The graph encodes the dependency between interacting actions of the components in the following manner. If a component has an input edge labeled a_i from a node $(\{a_1, \ldots, a_n\}, \alpha)$, then for a_i to occur in some interaction of $\{a_1, \ldots, a_n\}$ containing α it is necessary that all the actions labeling input edges of $(\{a_1, \ldots, a_n\}, \alpha)$ interact.

We call a circuit in the dependency graph *non trivial* if it contains more than one component node.

Example 2 (Producer/consumer) *Consider a producer providing data to two consumers. Interaction is by rendez-vous and takes place if at least one of the two consumers can get an item. The interaction model is described by* $C = \left\{\{put, get_1, get_2\}\right\}$ *and* $IC^+ = \{put \shortmid get_1, put \shortmid get_2, put \shortmid get_1 \shortmid get_2\}$. *The dependency graph is shown in figure 5.*

For a some action, let $en(a)$ be the predicate characterizing the states of $owner(a)$ in which a is enabled. For γ a non trivial circuit of the dependency graph, let

$$D(\gamma) = \bigwedge_{((c,\alpha),a,k) \in \gamma} \left(\left(\bigvee_{a' \in \alpha} \neg en(a') \right) \ \wedge \ en(a) \right) .$$

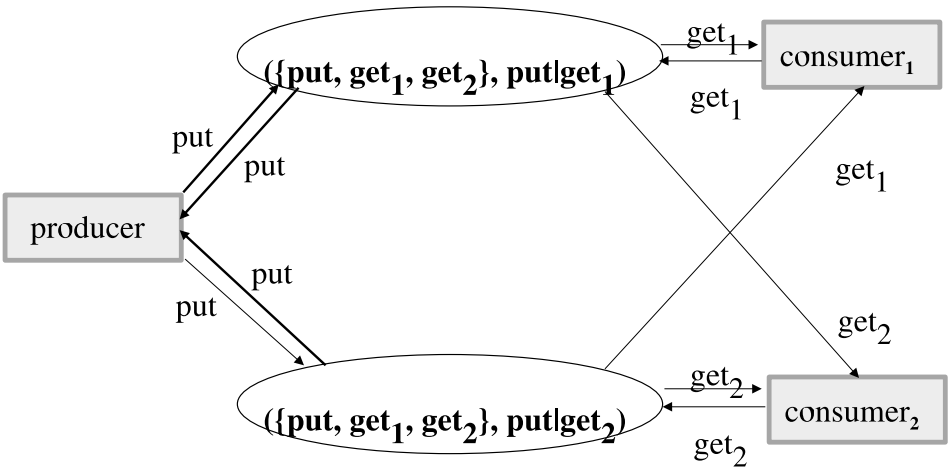

Fig. 5. Dependency graph for the producer/two consumer example.

Intuitively, $D(\gamma)$ is the predicate characterizing the product states for which all components in γ are cyclically waiting for each other and only non maximal incomplete interactions are possible. For a component k, let $inc(k)$ be the predicate characterizing the incomplete states of k. The circuit γ is called *feasible* if $\bigwedge_{k \in components(\gamma)} inc(k) \wedge D(\gamma) \neq false$.

Theorem 1 (Interaction safety) *A system is* interaction safe *if its dependency graph has a non-empty finite sub-graph G such that G contains all its predecessors, any component in G is deadlock-free, and there is no feasible non-trivial elementary circuit in G.*

Proof. Consider a state $q = (q_1, \ldots, q_n)$ of the system for which all components are in an incomplete state q_i from which only non-maximal incomplete actions are possible. Then each component k in G offers some non-maximal incomplete action a since it is deadlock-free. We consider the sub-graph G' of G that represents dependencies in the current state: G' has an edge from an interaction node (c, α) to a component node k if k is actually waiting for α in the current state; G' has the same edges from component to interaction nodes as G. G' has the same set of components as G since any component of G is awaiting at least one incomplete action.

If there is a non trivial circuit γ in G', then $\bigwedge_{k \in components(\gamma)} inc(k) \wedge D(\gamma) = false$ by hypothesis, and since all component states of q are incomplete, $D(\gamma)(q) = false$. Thus there is some edge $((c, \alpha), a, k)$ in γ such that $\bigwedge_{a' \in \alpha} en(a')(q) \ \vee \neg en(a)(q)$. By hypothesis, k is waiting for α, that is, $en(a)(q)$ holds. Therefore, $\bigwedge_{a' \in \alpha} en(a')(q)$, that is, interaction α is enabled.

If there is no non trivial circuit in G', suppose that G' is strongly connected (otherwise, take only the source strongly connected subgraph). Consider some interaction node (c, α) of G'. For any component k and action a_k such that $(k, a_k, (c, \alpha))$, the edge $((c, \alpha), a_k, k)$ is also in G' since G' is strongly connected,

and there is no non trivial circuit. By construction of G', k is waiting for α, and a_k is enabled in the current state. Therefore, α is enabled.

In both cases, at least one complete or maximal interaction is enabled, which means that any non-maximal incomplete interaction is disabled in $(B, \prec)$. ■

Example 3 (Producer/consumer) *For example 2, the dependency graph G is backward closed. Let $n_1 = (\{put, get_1, get_2\}, put \shortmid get_1)$ and $n_2 = (\{put, get_1, get_2\}, put \shortmid get_2)$. G contains three non-trivial elementary circuits $\gamma_1 = (producer, n_1, consumer_2, n_2)$, $\gamma_2 = (producer, n_2, consumer_1, n_1)$, and $\gamma_3 = (consumer_1, n_1, consumer_2, n_2)$. We obtain $D(\gamma_1) = false$, $D(\gamma_2) = false$, and*

$$\begin{aligned} D(\gamma_3) =& \Big(\neg en(put) \vee \neg en(get_1)\Big) \wedge en(get_2) \wedge \\ & \Big(\neg en(put) \vee \neg en(get_2)\Big) \wedge en(get_1) \\ =& \neg en(put) \wedge en(get_1) \wedge en(get_2) \end{aligned}$$

If the only incomplete action of producer *is* put *and all three components are deadlock-free, then $inc(producer) \wedge inc(consumer_1) \wedge inc(consumer_2) \wedge D(\gamma_3) = false$ and the system is interaction safe.*

Remark 3 *The condition of Theorem 1 that $D(\gamma) = false$ for any non-trivial circuit γ, is a generalization of the notion of* cooperativity *introduced in [11].*

4.3 Individual Deadlock-Freedom

In general, deadlock-freedom or even liveness of a system of components does not imply that any component is deadlock-free in any run of the system. Guaranteeing this stronger property is the objective of the following definitions and theorem.

Notation: We consider a set of components K and denote by $S[K'] = (B[K'], IM[K'])$ the sub-system of $S[K]$ with $IM[K'] = (IC[K'], IC[K']^+)$, for $K' \subseteq K$.

Definition 12 (Run) *A* run *of $S[K] = (B[K], IM[K])$ with $B[K] = (Q, IC[K], \rightarrow)$ is a maximal sequence of interactions $q_0 \stackrel{\alpha_0}{\rightarrow} q_1 \stackrel{\alpha_1}{\rightarrow} \ldots q_n \stackrel{\alpha_n}{\rightarrow} \ldots$ where $q_i \in Q$ and $\alpha_i \in IC[K]$ for all i.*

Definition 13 (Individual Deadlock-Freedom) *Given a system $S[K]$, a component $k \in K$ is* deadlock-free *in $S[K]$ if for any run σ of $S[K]$ and any prefix σ_n of σ, there exists a run σ' such that $\sigma_n\sigma'$ is a run of $S[K]$, and some interaction of σ' contains an interaction of k.*

Definition 14 (Controllable predecessors) *Let* $S[K'] = (B[K'], IM[K'])$ *with* $B[K'] = (Q, IC[K'], \rightarrow)$ *for* $K' \subseteq K$. *For* $X \subseteq Q$, *define* $pre(X) \subseteq Q$ *such that* $q \in pre(X)$ *if*

- *if* q *is complete then* $\exists q' \in Q\ \exists a \in IC[K']^+ \,.\, q \xrightarrow{a} q' \ \wedge\ q' \in X$;
- *if* q *is incomplete then* $\forall q' \in Q\ \forall a \in IC[K']^- \,.\, q \xrightarrow{a} q' \Rightarrow q' \in X$.

For $Q_0 \subseteq Q$ *we denote by* $PRE(Q_0)$ *the least solution of* $X = Q_0 \cup pre(X)$.

Clearly, $PRE(Q_0)$ is the fixed point of a monotonic functional. $PRE(Q_0)$ represents the set of the predecessors of Q_0 in the transition system such that from anyone of its states a state of Q_0 can be reached by appropriately choosing complete interactions. In this context, complete interactions can be characterized as controllable, as when they are enabled some interaction containing them can occur in the product system. On the contrary incomplete interactions can be considered as uncontrollable as their occurrence in the product depends on the state of the environment. Predicate transformers taking into account controllability have been studied in [21].

Definition 15 (Controllability) *Given a system* $S[K'] = (B[K'], IM[K'])$ *with* $B[K'] = (Q, IC[K'], \rightarrow)$ *for* $K' \subseteq K$, *we call* K' controllable *with respect to some interaction* $a \in IC[K']$ *if* $PRE(en(a)) = Q$.

K' *is* controllable *with respect to* $IM[K] = (IC[K], IC[K]^+)$ *if it is controllable with respect to any interaction* $\alpha \in IC[K']$ *such that there exists* $\alpha' \in IC[K]^-$ *with* $\alpha \iota \alpha' \in IC[K]$; *that is, there exists an interaction* α' *needing synchronization with* α.

Controllability of K' in $S[K]$ means that if progress in a component k of $S[K]$ requires some interaction from $S[K']$, the latter can be led to a state where synchronization is possible.

Theorem 2 (Individual Deadlock-Freedom) *A component* k *is deadlock-free in* $S[K]$ *with interaction model* $IM[K]$ *if the dependency graph of* $S[K]$ *has a finite sub-graph* G *such that (1)* k *is a node of* G, *(2)* G *contains all its predecessors, all its components are deadlock-free, and it has no feasible non-trivial elementary circuit, (3) all components in* G *are controllable with respect to* $IM[K]$, *and (4) for any* $c \in C[K]$, $|c| \leqslant 2$, *that is, all interactions between components are binary.*

Proof. Consider some product state in which k is blocked, and no complete or maximal interaction involving k is enabled. Suppose that k is waiting for some interaction α, and let $\bullet\alpha$ be the components participating in α. Any of them can either, by controllability, progress until its action synchronizing in α is enabled, or it is blocked itself. By theorem 1, some (direct or transitive)

predecessor of k can progress. Let $k_{i_1} \overset{\alpha_1}{\dashrightarrow} k_{i_2} \overset{\alpha_2}{\dashrightarrow} \ldots \overset{\alpha_{n-1}=\alpha}{\dashrightarrow} k_{i_n} = k$ be a chain of components where $k_i \overset{\alpha}{\dashrightarrow} k_j$ means that component k_j is in an incomplete state waiting for interaction α, and such that only k_{i_1} is able to progress. By (3), it can be led (by appropriately choosing some complete action, or by any incomplete action) towards a state where its action participating in α_1 is enabled, thus unblocking k_{i_2}. The same reasoning can now be applied to k_{i_3} and recursively descending the chain, until α becomes enabled. ■

5 Example: a Token-Ring Network

The following example models a token-ring network with three nodes. At any time exactly one network node possesses the token and can access the network. Figure 6 shows the architecture of the network. Each network node i consists of a network layer N_i and optionally, an application layer A_i. A network layer N_i in possession of the token can grant network access to the application layer A_i by interaction $grant_i | p_i$, until A_i frees the network in a $v_i | free_i$ interaction with N_i. The network layer can then pass the token to $N_{(i \mod 3)+1}$ by the interaction $\{pass_i, get_{(i \mod 3)+1}\}$. We require that all non-maximal interactions are incomplete and the maximal interactions are complete. Thus the interaction model IM has the set of connectors isomorphic to the set of complete interactions: $IC^{+} = C = \bigcup_{i=1,2,3}\{pass_i | get_{(i \mod 3)+1}, grant_i | p_i, v_i | free_i, req_i\}$. We first consider that there is no application layer A_3 using the network, and that this component is added later.

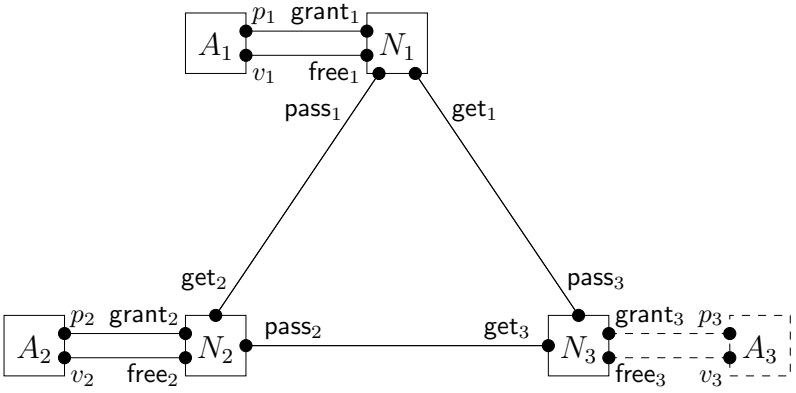

Fig. 6. Architecture of the token-ring.

Figure 7 shows the transition systems of one instance of the network layer N and the application layer A, respectively. After a get interaction, N can either pass on the token in a pass interaction, or grant access to the application layer (grant), wait for the latter to finish its transaction (free), and then pass on the token. The application layer can make a request to access the network (req), obtain access (p), and give back its right to access the network (v).

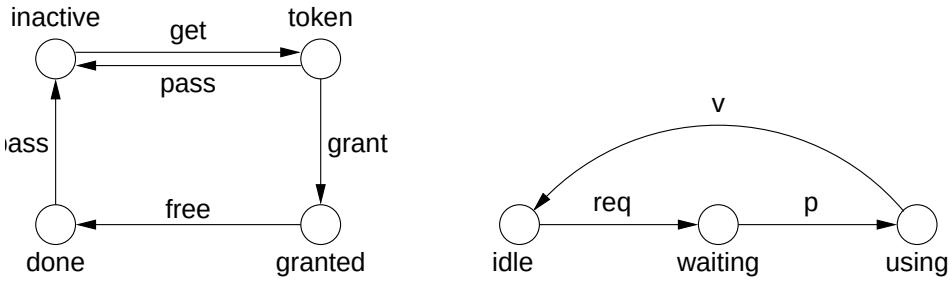

Fig. 7. Behavior of a network layer N (left) and an application layer A (right).

Let us check interaction safety of the network model. Figure 8 shows the dependency graph G. Labels on arcs have been omitted for the sake of readability; as connectors and maximal complete interactions coincide, only the latter are shown. There are four different types of non-trivial elementary circuits in G:

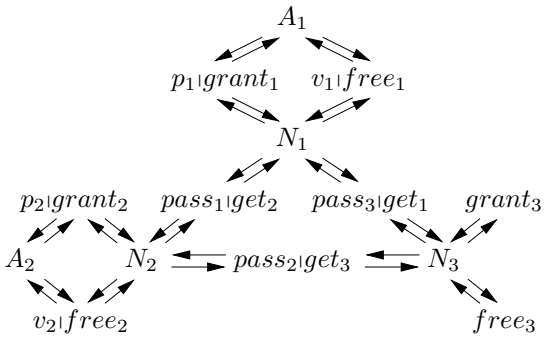

Fig. 8. Dependency graph of the token-ring.

$\gamma_1^i = (A_i, p_i|grant_i, N_i, v_i|free_i)$, $\gamma_2^i = (N_i, p_i|grant_i, A_i, v_i|free_i)$, for $i = 1, 2$
$\gamma_3 = (N_1, pass_1|get_2, N_2, pass_2|get_3, N_3, pass_3|get_1)$,
and $\gamma_4 = (N_3, pass_2|get_3, N_2, pass_2|get_1, N_1, pass_3|get_1)$. We have

$$
\begin{aligned}
D(\gamma_1^i) =& \neg en(p_i) \wedge en(grant_i) \wedge \neg en(free_i) \wedge en(v_i) = token_i \wedge using_i \\
D(\gamma_2^i) =& en(p_i) \wedge \neg en(grant_i) \wedge en(free_i) \wedge \neg en(v_i) = waiting_i \wedge granted_i \\
D(\gamma_3) =& \neg en(pass_1) \wedge en(get_2) \wedge \neg en(pass_2) \wedge en(get_3) \wedge \neg en(pass_3) \\
& \wedge en(get_3) = false \\
D(\gamma_4) =& en(pass_1) \wedge \neg en(get_2) \wedge en(pass_2) \wedge \neg en(get_3) \wedge en(pass_3) \\
& \wedge \neg en(get_3) = false
\end{aligned}
$$

Thus, Theorem 1 fails to prove interaction safety. Indeed, the system is not interaction safe, since there exist incomplete states $token_i \wedge using_i$ and $granted_i \wedge waiting_i$ in which network and application layer do not agree on the current state. However, it is easy to show that $U = \bigwedge_i (token_i \iff using_i)$ is an invariant of the global system. Therefore, initialized in a state in U the system always remains in a complete state, where interaction safety is guaranteed by Theorem 1.

It is easy to see that since the system is interaction safe, $U' = \neg N_1.inactive \wedge$

$N_2.inactive \wedge N_3.inactive \ \vee \ N_1.inactive \wedge \neg N_2.inactive \wedge N_3.inactive \ \vee \ N_1.inactive \wedge N_2.inactive \wedge \neg N_3.inactive$ is another invariant of the system, that is, there is always exactly one network layer component N_i possessing the token.

Let us see whether all components are deadlock-free in the system. The components A_i are controllable with respect to IM. The components N_i are controllable with respect to get_i, $grant_i$, and $pass_i$, but not with respect to $free_i$. By using knowledge about the overall system, it is possible to establish controllability of N_i with respect to $free_i$, and prove individual deadlock-freedom of all components.

Let us now add an application layer A_3 to our model. In $IM[\{N_1, N_2, N_3, A_1, A_2, A_3\}] = IM[\{N_1, N_2, N_3, A_1, A_2\}] \cup IM[\{A_3\}] \cup IM[\{N_1, N_2, N_3, A_1, A_2, A_3\}, \{A_3\}]$, $grant_3$ and $free_3$ are non-maximal incomplete interactions. By application of Theorem 1 to the modified dependency graph we show that the system remains interaction safe.

Given that all components are individually deadlock-free, a scheduler can be used to ensure fairness with respect to some or all components in the system. The scheduling policy modeled by the priority order $\{pass_i | get_{(i \mod 3)+1} \prec grant_i | p_i\}_{i=1,2,3}$ ensures requests of the application layer to be served before the token is passed on. The scheduled system is then obtained by restricting the system of interacting processes with these priority rules. As the components are deadlock-free and the composition of the priority order induced by the interaction model with the priority order of the scheduler is again a priority order, the obtained model is deadlock-free by Propositions 5 and 4.

6 Discussion

The paper proposes a framework for component composition encompassing heterogeneous interaction.

The framework uses a single abstract associative and commutative composition operator for layered components. Component layering seems to be instrumental for defining such an operator. Existing formalisms combine at the same level composition of both behavior and interaction constrains. Layered models allow separation of concerns by composing behaviors and interaction constraints separately. This makes the definition of a single associative operator technically possible.

Interaction models describe architectural constraints on component behavior. Connectors relate interacting actions of different components. They naturally

define the set of interactions in a system. The distinction between complete and incomplete interactions is essential for the unification of existing interaction mechanisms, e.g. unification of symmetric and asymmetric interaction. The notion of interaction safety characterizes correctness of a model with respect to strict synchronization requirements. Such requirements are implicit in existing formalisms, because they adopt specific interaction models, e.g. in CSP, all the synchronizing actions are incomplete. Interaction models allow incremental description of architecture constraints for heterogeneous interaction.

Flexible composition is just one possible implementation of the abstract composition operator, using priorities. An advantage over standard approaches, based on the use of restriction instead of priorities, is that restriction is adequate only for strict synchronization. It cannot express maximal progress for complete actions. Furthermore, flexible composition preserves deadlock-freedom. Transition systems with priorities prove to be powerful tools for incremental modeling. They are a very simple semantic model amenable to correctness by construction.

An essential correctness requirement for system models is interaction safety. Theorem 1 provides sufficient conditions for a system to be interaction safe from properties of its interaction model and of its components. It uses dependency graphs to represent synchronization constraints induced by interaction models. It would be interesting to investigate whether analysis techniques specific to dependency graphs can be adapted to the analysis of systems of interacting components. Notice that the same theorem can be used to guarantee global deadlock-freedom of a system obtained by using ordinary (non flexible) composition operators. That is, when priority orders are replaced by restriction in the product. In fact, restriction is used to remove non-maximal incomplete interactions. Proving that such interactions will never occur in a globally deadlock-free system amounts to proving that the system remains deadlock-free when these interactions are removed.

Finally, Theorem 2 establishes links between a notion of component controllability, with respect to an interaction model and individual deadlock-freedom. As shown in the previous example, this result can be extended to take into account the dynamics of a component's environment.

The framework presented is part of a lasting research effort for theory and methods guaranteeing system correctness by construction. It can be enriched to model components with interfaces and a notion of abstraction. It also needs further validation by examples and case studies. Nevertheless, its key features, such as combination of behavior and priorities and the resulting advantages in composability and compositionality, have already been positively assessed in some non-trivial applications [18] and [12].

References

[1] K. Altisen, G. Gössler, and J. Sifakis. Scheduler modeling based on the controller synthesis paradigm. *Journal of Real-Time Systems, special issue on "control-theoretical approaches to real-time computing"*, 23(1/2):55–84, 2002.

[2] F. Balarin, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, M. Sgroi, and Y. Watanabe. *Modeling and Designing Heterogeneous Systems*, volume 2549 of *LNCS*, pages 228–273. Springer-Verlag, 2002.

[3] A. Benveniste, P. LeGuernic, and Ch. Jacquemot. Synchronous programming with events and relations: the SIGNAL language and its semantics. *Science of Computer Programming*, 16:103–149, 1991.

[4] J.A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *TCS*, 37(1):77–121, 1985.

[5] G. Berry and G. Gonthier. The ESTEREL synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.

[6] S. Bornot, G. Gössler, and J. Sifakis. On the construction of live timed systems. In S. Graf and M. Schwartzbach, editors, *Proc. TACAS'00*, volume 1785 of *LNCS*, pages 109–126. Springer-Verlag, 2000.

[7] S. Bornot and J. Sifakis. An algebraic framework for urgency. *Information and Computation*, 163:172–202, 2000.

[8] L. de Alfaro and T.A. Henzinger. Interface theories for component-based design. In T.A. Henzinger and C. M. Kirsch, editors, *Proc. EMSOFT'01*, volume 2211 of *LNCS*, pages 148–165. Springer-Verlag, 2001.

[9] W.-P. de Roever, F. de Boer, U. Hannemann, J. Hooman, Y. Lakhnech, M. Poel, and J. Zwiers. *Concurrency Verification: Introduction to Compositonal and Noncompositional Methods*. Cambridge University Press, 2001.

[10] W.-P. de Roever, H. Langmaack, and A. Pnueli, editors. *Compositionality: The Significant Difference*, volume 1536 of *LNCS*. Springer-Verlag, 1997.

[11] G. Gössler and J. Sifakis. Composition for component-based modeling. In *proc. FMCO'02*, volume 2852 of *LNCS*. Springer-Verlag, 2003.

[12] S. Graf, I. Ober, and I. Ober. Model checking of uml models via a mapping to communicating extended timed automata. In S. Graf and L. Mounier, editors, *Proc. SPIN'04*, volume 2989 of *LNCS*. Springer-Verlag, 2004.

[13] OMG Working Group. Response to the omg rfp for schedulability, performance, and time. Technical Report ad/2001-06-14, OMG, June 2001.

[14] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[15] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[16] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[17] ITU-T. Recommendation Z.100. Specification and Design Language (SDL). Technical Report Z-100, International Telecommunication Union — Standardization Sector, Geneva, 1999.

[18] C. Kloukinas, C. Nakhli, and S. Yovine. A methodology and tool support for generating scheduled native code for real-time java applications. In R. Alur and I. Lee, editors, *Proc. EMSOFT'03*, volume 2855 of *LNCS*, pages 274–289, 2003.

[19] L. Lamport. Specifying concurrent program modules. *ACM Trans. on Programming Languages and Systems*, 5:190–222, 1983.

[20] E.A. Lee et al. Overview of the Ptolemy project. Technical Report UCB/ERL M01/11, University of California at Berkeley, 2001.

[21] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In E.W. Mayr and C. Puech, editors, *STACS'95*, volume 900 of *LNCS*, pages 229–242. Springer-Verlag, 1995.

[22] F. Maraninchi. Operational and compositional semantics of synchronous automaton compositions. In *proc. CONCUR*, volume 630 of *LNCS*. Springer-Verlag, 1992.

[23] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25(3):267–310, 1983.

[24] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[25] SystemC. `http://www.systemc.org`.