A Framework for Component-based Construction
Extended Abstract

Joseph Sifakis
VERIMAG Laboratory
Centre Equation
2 avenue de Vignate, 38610 GIERES, France
Joseph.Sifakis@imag.fr

Abstract

We present an overview of results developed mainly at Verimag, by the author and his colleagues, on a framework for component-based construction, characterized by the following:

• The behavior of atomic components is represented by transition systems;

• Components are built from a set of atomic components by using "glue" operators;

• For each component, it is possible to separate its behavior from its structure, due to specific properties of glue operators.

We show an instance of this framework, which combines two independent classes of glue operators, Interaction Models and Priorities.

The combination of interaction models and priorities is expressive enough to encompass heterogeneous interaction and execution. We show that separation between behavior and structure is instrumental for correctness-by-construction. Finally, we discuss new research problems related to a structure-dependent notion of expressiveness.

1. Introduction and Key issues

1.1. Brief Overview

A central idea in systems engineering is that complex systems are built by assembling components (building blocks). Components are systems characterized by their interface, an abstraction that is adequate for composition and re-use. It is possible to get large components by "gluing" together simpler ones. "Gluing" can be considered as an operation on sets of components.

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 seen significant development, especially through 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. There exists a large body of literature dealing with components and their use for different purposes and in different contexts.

The following deal, one way or another, with issues related to component-based engineering:

- Software Design Description Languages such as [6, 5] and Architecture Description Languages focusing on non-functional aspects such as [18, 1];
- System modeling languages such as UML [17], as well as languages and notations specific to tools such as Simulink/Stateflow, SystemC [15], Metropolis[3], Ptolemy [13], IF-toolset [4];
- Coordination languages extensions of programming languages such as Linda, Javaspaces, TSpaces, Concurrent Fortran;
- Middleware standards such as IDL, Corba, Javabeans, .NET;
- Software development environments such as PCTE, SWbus, Softbench, Eclipse;
- Theoretical frameworks based on process algebras e.g., the Pi-Calculus [14] or based on automata e.g., [16].

An assessment of the above is beyond the scope of the paper. Nevertheless, each fails to satisfy at least one of the following:

- Be founded on rigorous semantics and provide concepts supporting separation of concerns e.g., decou-

Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05)
0-7695-2435-4/05 $20.00 © 2005 IEEE

pling between behavior and interaction. This is particularly the case for modeling, as well as for middleware and software development standards. These use ad hoc mechanisms for building systems from components and offer syntax-level concepts only.

- Encompass heterogeneous description, as they support only specific interaction mechanisms and computation models. For instance, software design frameworks are based on interaction by method call and do not allow direct modeling of synchronous interaction mechanisms. On the contrary, other frameworks such as SystemC and Matlab/Simulink have built-in mechanisms for synchronous execution, and are not adequate for describing asynchronous systems.

- Encompass the description of timing and resource management, which are essential for non-functional properties. For instance, standards such as UML or AADL offer only syntactic sugar for time and scheduling policies. The lack of adequate semantic frameworks does not allow checking for inconsistency in timing requirements, or the meaningful composition of scheduling policies.

- Consider architectures as first class entities. For example, existing theoretical frameworks are too low-level, since they only emphasize behavioral aspects.

1.2. System Construction

The system construction problem can be formulated as follows.

Build a component C satisfying a given property P, from a given set of atomic components $\mathbf{C_a}$ and a set GL of operators on these components.

The component C to be constructed, can be considered to be a term of the algebra generated from $\mathbf{C_a}$ and GL. There exist only a few algebraic frameworks for formalizing system construction problems, such as boolean algebra for logical circuits, Hoare logic for programs, process algebras with modal logic. At the same time, most of the existing results on protocols and distributed algorithms define solutions to construction problems for meeting specific properties in component-based systems. For example, a token-ring protocol guarantees mutual exclusion between interacting components. As a rule, construction problems can be formulated as highly intractable synthesis problems.

Existing frameworks for studying component-based construction fail to adequately treat at least one of the following important requirements:

- Encompass heterogeneous composition to ensure interoperability of components, as explained in the next sub-section 1.3;

- Provide results guaranteeing correctness-by-construction for essential system properties such as deadlock-freedom, progress and liveness, in order to minimize a posteriori validation, as explained in 2.2

1.3. Heterogeneity

System designers deal with a large variety of components, each having different characteristics. A central problem is "meaningful" composition of such components to ensure that they interoperate correctly. We need semantic frameworks encompassing heterogeneous composition. There exist three specific sources of heterogeneity: interaction, execution and abstraction.

Heterogeneity of Interaction Interactions are combinations of actions performed by system components to achieve a desired global behavior. Interactions can be *atomic* or *non atomic*. For atomic interactions, the state change induced in the participating components cannot be altered through interference with other interactions. As a rule, synchronous languages and hardware description languages use atomic interactions. On the contrary, languages with buffered communication (SDL) or multi-threaded languages (Java, UML), generally use non-atomic interactions.

Both types of interactions may involve *strong* or *weak* synchronization. Interactions involving strong synchronization can occur only if all the participating components agree, for instance interaction via rendezvous involves strong synchronization (and is atomic). Interactions using weak synchronization are asymmetric and require only the participation of an initiating action, that may synchronize with other actions, such as "outputs" in synchronous languages.

Heterogeneity of Execution Currently, there exists no formalism encompassing both synchronous and asynchronous execution. Synchronous execution is typically used in hardware, synchronous languages and time-triggered systems. It considers that a system's execution is a sequence of global steps. It assumes synchrony, meaning that the 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 can move forward.

Asynchronous execution does not use any notion of a global computation step. It has been adopted in most distributed system description languages 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 and tasks.

Heterogeneity of Abstraction System development involves the use of languages, models and physical implementations representing a system and its components at different abstraction levels. For heterogeneity, a key abstraction is the one relating an application software to its implementation on a given platform.

Application software is *untimed* in the sense that it abstracts out physical time. The only references to physical time are time parameters of real-time statements, such as timeouts and watchdogs. The expiration of watchdogs or timeouts is treated at the semantic level as an external event.

An application software running on a given platform, is a *timed* system. The set of its state variables includes not only the variables of the application software but also all the variables needed to characterize its dynamic behavior such as time, quantity of resources e.g., memory and power.

We need abstractions and theory relating application software to its implementations. In particular, such abstractions should guarantee the preservation of functional properties.

In section 2, we present a framework intented to meet the above requirements for the following particular instance of the construction problem:

For a given a set of atomic deadlock-free components from $\mathbf{C_a}$*, find "glue" such that the resulting system is deadlock-free and meets a given safety property* P*.*

In subsection 2.1, we introduce *glue* operators which transform sets of components into new components. We present an overview of results about two independent classes of such operators and their combined use.

- Interaction models providing a general mechanism for modeling the interactions between a set of components.
- Priorities providing a general mechanism for restricting the behavior of a set of components by preserving deadlock-freedom.

In subsection 2.2, we describe an approach for correctness-by-construction, and provide an overview of existing results. Finally, in section 3, we discuss open problems and future research directions.

2. The Framework

2.1. Glue Operators and their Properties

Components For a given vocabulary of actions A, we denote by $\mathbf{B}$ a set of transition systems $\mathbf{B} = \{B_i\}_i$ with disjoint sets of of actions $A_i \subset A$. A transition system B_i is defined as a set of transitions of the form (s, a, s') where $a \in A_i$ and $s, s' \in S_i$, the set of the states of B_i.

The set of the components $\mathbf{C}$ defined from a given set of transition systems $\mathbf{B}$ and a set of *glue operators* $GL = \{gl_j\}_j$, is the set of the terms of the form $gl(U)$ where U is any non empty set such that $U \subseteq \mathbf{C} \cup \mathbf{B}$.

Furthermore, we require for terms representing components $gl(U) = glue(\{U_1, .., U_n\})$, to have disjoint sets $actions(U_i)$, where $actions$ is a function defined by:

- $actions(glue(\{U_1, .., U_n\})) = actions(U_1) \cup \ldots \cup actions(U_n)$
- $actions(B_i) = A_i$.

The definition of glue operators for sets makes simpler the treatment of their specific properties given below.

To simplify notation, we write $gl\{U_1, .., U_n\}$ where the U_i are components or transition systems.

Components are characterized by their *behavior* and their *structure*. A term of the form $gl\{B_1, .., B_n\}$ represents a component in "flattened" form where structure is separated from behavior.

Glue operators are characterized by the possibility to put any component represented as a term of $\mathbf{C}$ into flattened form. This is achieved by,

- Introducing an idempotent *composition operation* $\oplus$ on GL such that $(GL, \oplus)$ is a commutative monoid.
- Assuming that $\mathbf{C}$ is equipped with a congruence relation $\cong$ such that the following characteristic property is satisfied:

For any term of the form $t' = gl(U \cup gl'(U'))$ *there exists a term of the form* $t'' = gl \oplus gl''(U \cup U')$*, such that* $t' \cong t''$*.*

This property allows separation between behavior and structure, by reducing any component into a term of the form $gl\{B_1, \ldots, B_n\}$. Such a separation is instrumental for considering structure as first class entity.

The behavior of components is defined by considering that glue operators transform sets of transition systems into transition systems e.g., by using operational semantics.

We present results related to two classes of glue operators: interaction models and priorities.

Interaction Models and their properties Consider as in the previous paragraph, a vocabulary of actions A and a set of transition systems $\mathbf{B}$. Interaction models are a class of glue operators characterizing the interactions of components.

An *interaction* on A is any non empty subset of A. Interactions correspond to sets of actions which may synchro-

nize. Two interactions are non comparable if none of them is contained in the other.

An *interaction model* im on A, is a pair $im = (\Gamma, \Delta)$ where:

• Γ is a set of non comparable interactions of A called *connectors* of im;

• Δ is a set of non comparable interactions of A such that any interaction of Δ is contained in some interaction of Γ. Δ is the set of the *minimal complete interactions* of im.

The interactions α defined by an interaction model im satisfy the following:
• they are contained in some connector of Γ, that is $\exists \gamma \in \Gamma$. $\alpha \subseteq \gamma$
• either they contain some interaction of Δ or they are maximal, that is $\exists \delta \in \Delta$. $\delta \subseteq \alpha$ or $\exists \gamma \in \Gamma$. $\alpha = \gamma$.

Interaction models provide a powerful means to describe both strong and weak synchronization.

For strong synchronization, only the interactions corresponding to connectors are possible. For example, the interaction model for$\Gamma = \{\{a_1, a_2, a_3\}\}$ and $\Delta = \emptyset$ contains only the interaction $\{a_1, a_2, a_3\}$ which means that the actions a_1, a_2, a_3 must synchronize.

For weak (asymmetric) synchronization, minimal possible interactions are defined by elements of Δ. For example, the interaction model with $\Gamma = \{\{a_1, a_2, a_3\}\}$ and $\Delta = \{\{a_1\}\}$ consists of all the interactions that contain a_1. It can be used to characterize asymmetric synchronization where for example, an output a_1 is broadcast to two inputs a_2 and a_3.

To define the operational semantics of terms, we consider that interaction models are operators on transition systems. The term $im\{B_1,\ldots, B_n\}$ defines a transition system such that if there exists a set of indices J

$\exists \{a_j\}_{j\in J} \in im$ and $\forall j \in J$. $(s_j, a_j, s'_j) \in B_j$, then $((s_1,\ldots, s_n), \{a_j\}_{j\in J}, (s'_1,\ldots, s'_n)) \in im\{B_1,\ldots, B_n\}$
where $\forall i \notin J$. $s_i = s'_i$.

Given interaction models $im_i = (\Gamma_i, \Delta_i)$ for $i = 1, 2$, the operation $\oplus$ is defined by

$im_1 \oplus im_2 = (\Gamma, \Delta)$ such that

$\Gamma = max(\Gamma_1 \cup \Gamma_2)$ and $\Delta = min(\Delta_1 \cup \Delta_2)$ where max and min are respectively the functions giving the set of the maximal and minimal elements of their arguments.

Clearly, the operation is associative. The set of the connectors of the resulting interaction model contains all the connectors except those contained in some connector of the union of the connector sets. The dual operation is applied for minimal complete interactions.

In the rest of this paragraph, we present properties of interaction models and their use for incremental construction of systems.

Let $im = (\Gamma, \Delta)$ by an interaction model on A. It is possible to find decompositions of im with respect to a set of disjoint action vocabularies $\{A_1,\ldots, A_n\}$ such that $A_i \subseteq A$, for $1 \leq i \leq n$. We need the following two definitions:

• The interaction model $im[A_i] = (\Gamma_i, \Delta_i)$ includes exactly all the interactions of im involving only actions from A_i. It is defined by

$$\Gamma_i = \{\gamma' | \exists \gamma \in \Gamma. \gamma' = \gamma \cap A_i\},$$
$$\Delta_i = \{\delta \in \Delta | \exists \gamma \in \Gamma_i.\ \delta \subseteq \gamma\}.$$

• The interaction model $im[A_1,\ldots, A_n] = (\Gamma', \Delta')$ contains all the interactions of $im[A_1 \cup \ldots \cup A_n]$ which are not interactions of $im[A']$ for any set A' union of $n-1$ elements of $\{A_1,\ldots, A_n\}$. It is defined by

$$\Gamma' = \{\gamma' = \cup_{1\leq i\leq n}\gamma_i | \gamma_i \in \Gamma_i \wedge \exists \gamma \in \Gamma\ .\ \gamma' = \gamma \cap \cup_{1\leq i\leq n} A_i\}$$
$$\Delta' = \{\delta' \in \Delta | \exists \gamma' \in \Gamma'.\ \delta' \subseteq \gamma'\}$$

We provide instances of general results presented in [10, 8].

• $im[A_1 \cup A_2] = im[A_1] \oplus im[A_2] \oplus im[A_1, A_2]$

This equality gives the interaction model on the union of two action vocabularies as a the decomposition of the interaction models on each action vocabulary and $im[A_1, A_2]$. The latter includes the connectors of $im[A_1 \cup A_2]$ which are obtained by "gluing together" connectors from $im[A_1]$ and $im[A_2]$.

• The above equality can be generalized to obtain a decomposition of im with respect to the set $\{A_1,\ldots, A_n\}$ by using the property

$im[A_1 \cup A_2, A_3] = im[A_1, A_3] \oplus im[A_2, A_3] \oplus im[A_1, A_2, A_3]$.

For instance, the decomposition with respect to $\{A_1, A_2, A_3\}$ is given by:

$im[A_1 \cup A_2 \cup A_3] = im[A_1] \oplus im[A_2] \oplus im[A_3] \oplus im[A_1, A_2] \oplus im[A_2, A_3] \oplus im[A_1, A_3] \oplus im[A_1, A_2, A_3]$.

These results allow incremental construction of components by using binary glue operators which play the role of parallel composition operators, as shown below.

Consider a component $im(U)$ obtained by gluing together elements of U such that $actions(U) = A$ and $\{U_1,\ldots, U_n\}$ is a set of subsets of U with disjoint action vocabularies $A_i = actions(U_i)$ for $1 \leq i \leq n$.

• $im[A_1, A_2]\{im[A_1](U_1), im[A_2](U_2)\} = im[A_1, A_2] \oplus im[A_1] \oplus im[A_1](U_1 \cup U_2) = im[A_1 \cup A_2](U_1 \cup U_2)$.

The operator $im[A_1, A_2]$ can be considered as a parallel composition operator between $im[A_1](U_1)$ and $im[A_2](U_2)$ parameterized by the interaction model including all the interactions involving actions from both A_1 and A_2

The results on the decomposition of interaction models can be used to show that $im(U)$ can be constructed from its atomic components by using only binary glue operators depending on the chosen decomposition path. For example, if $\{U_1, U_2, U_3\}$ is a partition of U, $im(U)$ can be obtained by applying $im[A_1 \cup A_2, A_3]$ to $\{im[A_1, A_2]\{im[A_1](U_1), im[A_2](U_2)\}, im[A_3](U_3))\}$ or by applying $im[A_1, A_2 \cup A_3]$ to $\{im[A_1](U_1), im[A_2, A_3]\{im[A_2](U_2), im[A_3](B_3)\}\}$.

Priority Operators and their Properties Priorities are a class of glue operators PR restricting the behavior of components. For $pr \in PR$, the meaning of $pr\{U_1, \ldots, U_n\}$ is defined as follows:

• Assume that the behaviors of U_i, are represented by a transition system B_i on a vocabulary A_i with set of states S_i. Let B be the product transition system on the vocabulary $A = \cup_{1 \leq i \leq n} A_i$ with set of states $S = \times_{1 \leq i \leq n} S_i$. That is $(s_i, a_i, s_i') \in S_i$ for some i, $1 \leq i \leq n$, implies $((s_1, .., s_i, .., s_n), a_i, (s_1, .., s_i', .., s_n))$.

• A *priority* pr is a function associating with each state $s \in S$ a strict partial order $pr(s)$ on A.

The behavior of $pr\{U_1, \ldots, U_n\}$ is represented by a transition system B' such that $(s, a', s') \in B'$ if $(s, a', s') \in B$ and $\nexists a'', s''.\ (s, a'', s'') \in B$ and $a' pr(s) a''$.

As $B' \subseteq B$, priority operators simply restrict the behavior of the components of their arguments.

Given two priority operators pr_1 and pr_2, we define a composition operator $\oplus$ such that for any state $s \in S$ $pr_1(s) \oplus pr_2(s)$ is the least partial order such that $pr_1(s) \cup pr_2(s) \subseteq pr_1(s) \oplus pr_2(s)$, if such a partial order exists.

We have extensively studied the use of priority operators as a means to restrict the behavior of a set of components, so as to meet a given property P. In [2] we show that priorities provide a general framework for modeling and composing scheduling policies. In [9] we generalize these results in the following manner.

Given a system consisting of a set of interacting components and a global safety property P of this system, it is possible to define controllers (e.g., by fixpoint characterization) which interact with the system so that the controlled system satisfies P and is deadlock-free. We have characterized the behavior of the controlled system by a class of functions that restrict the enabling conditions of the interactions of the initial system. The main result is that priority operators characterize exactly the class of the restrictions induced by safe and deadlock-free controllers. That is, they are expressive enough to describe any solution to the system construction problem for safety and deadlock-freedom. A consequence of this result is that mutual exclusion properties can be modeled by using priorities, if they preserve deadlock-freedom.

Another class of results deals with composability of restrictions induced by priorities. It can be shown that in general, $pr_1\{pr_2\{B\}\} \neq pr_2\{pr_1\{B\}\}$, that is the transition system obtained by successive application of pr_1 and pr_2, depends on the order of application.

Furthermore, $pr_1 \oplus pr_2\{B\} \subseteq pr_1\{pr_2\{B\}\}$ and equality holds equality holds if and only if $pr_1 \cup pr_2 = pr_1 \oplus pr_2$.

As we consider priorities to be glue operators, we use the axiom $pr_1 \oplus pr_2\{B\} = pr_1\{pr_2\{B\}\}$ by assuming that only flattened terms represent transition systems.

The Layered Component Model

We have developed a modeling methodology which considers that components consist of three distinct layers [10]. The bottom layer includes a set of transition systems modeling behavior. The intermediate layer is an interaction model and the upper layer is a priority.

Layered components can be represented by using composite glue operators of the form $<pr, im>$. If U is a set of components we take $<pr, im>(U) = pr(im(U))$. That is, pr is used to filter the interactions of $im(U)$. The principle of layered construction is applied by the simulation kernel in the IF toolset [4].

We extend the composition operation $\oplus$:
$<pr_1, im_1> \oplus <pr_2, im_2> = <pr_1 \oplus pr_2, im_1 \oplus im_2>$.

The combined use of interaction models and priorities confers numerous advantages. It can be shown through examples, using only interaction models to describe components' coordination may lead to cumbersome solutions. There are coordination problems for which priorities are more appropriate e.g., scheduling problems, while for other problems the use of interactions is instrumental for getting simple solutions e.g., data flow problems. Another advantage is the existence of results guaranteeing, by construction, generic properties (see next paragraph).

2.2. Correctness-by-Construction

In principle, component-based frameworks should allow inferring system properties from properties of their structure. Currently, most of the existing validation techniques e.g., model-checking, need the construction of global models. Other techniques exist, using properties of a system's structure e.g., axiomatic verification techniques [12] and

assume-guarantee techniques [11]. These are developed for correctness with respect to general properties. They are of limited practical interest, since they make use of inference rules whose premises involve "oracles", unknown predicates, which in general, are non-computable.

We need theory, methods and tools for establishing, by construction, overall system correctness from component properties. The idea of building systems that are correct-by-construction is much more common than one would believe. All proven algorithms, protocols and architectures provide recipes for building correct systems. Nevertheless, these are for very specific properties such as lossless message transmission, clock synchronization etc. and they involve more or less sophisticated construction principles. We need lightweight theory for establishing generic system properties such as deadlock-freedom or progress of integrated components.

Two types of rules are necessary for correctness-by-construction:

- *Compositionality rules* which allow inferring global system properties from component properties. That is, if for $i = 1,\ldots,n$ a set of components U_i satisfy properties P_i, then it is possible to guarantee that the component $gl(U_1,\ldots,U_n)$ satisfies some property $\tilde{gl}(U_1,\ldots,U_n)$ where $\tilde{gl}$ is an operator on properties depending on gl.

- *Composability rules* which allow inferring that a component's properties are not affected when its structure is modified. That is, if components $gl\{U_1,\ldots,U_n\}$ and $gl'\{U_1,\ldots,U_n\}$ respectively satisfy the properties P and P', then the component $gl \oplus gl'\{U_1,\ldots,U_n\}$ satisfies $P \wedge P'$. Composability means stability of component properties across integration. Property instability phenomena are currently poorly understood e.g., feature interaction in telecommunications, or non composability of scheduling algorithms.

In [7], we provide sufficient conditions for deadlock-freedom of a layered component of the form $< pr, im > \{B_1,\ldots,B_n\}$ from deadlock-free atomic components. We assume that for the behavior B_i of each atomic component, a deadlock-free invariant is given. That is, a set of states from which the component can be blocked only because of its environment. The conditions relate deadlock-free invariants of the atomic components to the enabling conditions of the interactions of $<pr, im>$. These are computed by analysis of the structure of the operator $<pr, im>$, represented as a bipartite dependency graph relating actions of atomic components to interactions of im.

We also provide composability results, sufficient conditions for individual deadlock-freedom and liveness of the atomic components which are integrated in $< pr, im > \{B_1,\ldots,B_n\}$. The atomic component $im\{B_i\}$ is individually deadlock-free (respectively live) if it is always possible (respectively always inevitable) to perform some action of A_i in $< pr, im > \{B_1,\ldots,B_n\}$. The conditions require that atomic components have a set of states from which they are both deadlock-free and it is possible (respectively inevitable) to enable interactions with other components.

3. Perspectives for Future Work

The framework developed and the underlying system construction methodology have been successfully applied. We believe that the existing results and their application provide evidence that they adequately address many key issues in component-based systems modeling. We plan to continue this work in the following directions.

- Provide an algebraic formalization of *component algebras*. As suggested in 2.1, these can be defined as structures of the form $CA = (\mathbf{B}, GL, \oplus, \cong)$ where $\mathbf{B}$ is a set of atomic behaviors, $(GL, \oplus)$ is a commutative monoid and $\cong$ is compatible with the structure-behavior separation property described in 2.1. Such a formalization should be based on sufficiently general semantics for glue operators, and focus on properties relating the operator $\oplus$ and the relation $\cong$.

- Find classes of glue operators different from interaction models and priorities. For instance, one can consider glue operators expressing mutual exclusion constraints e.g., characterized as sets of mutually exclusive actions. We have shown that such constraints can be expressed for deadlock-free systems, by using priorities.

- An interesting theoretical question is the definition of a notion of *expressiveness* for component algebras taking into account component structure. Consider two component algebras $CA_i = (\mathbf{B}, GL_i, \oplus_i, \cong_i)$ for $i = 1, 2$ with the same set of atomic behaviors.

  We say CA_1 is more expressive than CA_2, if for any global property P on the product behavior of a set of atomic behaviors $\{B_1,\ldots,B_n\}$ from $\mathbf{B}$, for each component $gl_2\{B_1,\ldots,B_n\} \in CA_2$ satisfying P there exists a component $gl_1\{B_1,\ldots,B_n\} \in CA_1$ satisfying P. Notice the difference with existing notions of expressiveness which either completely ignore structure e.g. by considering that all the languages describing finite state systems are equivalent or compare languages of terms where separation between structure and behavior seems problematic. This is probably the case for process algebras which combine parallel composition operators with hiding and restriction. For instance, the expressive equivalence between SCCS and

CCS, may not be valid with respect this new notion of expressiveness.

The proposed notion of expressiveness can provide a basis for comparing frameworks for the construction of distributed systems. The property P can be the specification of a coordination problem between components e.g., mutual exclusion, clock synchronization etc. Typical questions to be addressed are whether a problem can be solved by using a given component algebra. For instance, is a component algebra built only from binary connectors with weak synchronization as expressive as component algebras using $n - ary$ connectors with rendezvous? Which coordination problems can be solved by using only static priorities (and no interactions at all)?

- Study techniques for correctness-by-construction especially to make their application more incremental and general. Their extension in order to deal with any safety property is in principle, possible. We need tractable techniques adapted to specific classes of properties e.g., mutual exclusion.

- Compare the layered component framework against existing ones in Ptolemy and Metropolis. A prototype implementation for this framework is currently under evaluation at Verimag. The prototype supports any kind of interaction model and dynamic priorities. The objective is to show that the framework can serve as a general "semantic middleware" for the execution of system description languages.

Aknowledgements: Most of the technical results have been developed in collaboration with Gregor Gößler from INRIA. Marius Bozga and Susanne Graf contributed through constructive discussions.

References

[1] R. Allen, S. Vestal, D. Cornhill, and B. Lewis. Using an architecture description language for quantitative analysis of real-time systems. In *Workshop on Software and Performance*, pages 203–210, 2002.

[2] K. Altisen, G. Gößler, and J. Sifakis. Scheduler modeling based on the controller synthesis paradigm. *Journal of Real-Time Systems, special issue on Control Approaches to Real-Time Computing*, 23(1-2):55–84, 2002.

[3] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: An integrated electronic system design environment. *IEEE Computer*, Apr 2003.

[4] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis. The IF toolset. In *4th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Real Time, SFM-04:RT, Bologna, Sept. 2004*, LNCS Tutorials, Springer, 2004.

[5] R. Bruni, J. L. Fiadeiro, I. Lanese, A. Lopes, and U. Montanari. New insights on architectural connectors. In *IFIP TCS*, pages 367–380, 2004.

[6] D. Garlan and B. R. Schmerl. Using architectural models at runtime: Research challenges. In *EWSA*, pages 200–205, 2004.

[7] G. Gößler and J. Sifakis. Component-based construction of deadlock-free systems: Extended abstract. In *FSTTCS*, pages 420–433, 2003.

[8] G. Gössler and J. Sifakis. Composition for component-based modeling. In *1st Symposium on Formal Methods for Components and Objects, revised lectures*, volume 2852 of *LNCS Tutorials*, 2003.

[9] G. Gössler and J. Sifakis. Priority systems. In *proceedings of FMCO'03*, LNCS 3188, pages 314–329, 2004.

[10] G. Gössler and J. Sifakis. Composition for component-based modeling. *Sci. Comput. Program.*, 55(1-3):161–183, 2005.

[11] T. A. Henzinger, S. Qadeer, and S. K. Rajamani. Decomposing refinement proofs using assume-guarantee reasoning. In *ICCAD*, pages 245–252, 2000.

[12] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

[13] E. Lee. Overview of the Ptolemy Project, Technical Memorandum UCB/ERL M03/25, July 2003.

[14] R. Milner. The pi calculus and its applications (keynote address). In *IJCSLP*, pages 3–4, 1998.

[15] W. Mueller, J. Ruf, D. Hofmann, J. Gerlach, T. Kropf, and W. Rosenstiehl. The simulation semantics of systemc, 2001.

[16] A. Ray and R. Cleaveland. Architectural interaction diagrams: Aids for system modeling. In *ICSE*, pages 396–407, 2003.

[17] B. Selic. Tutorial: An overview of uml 2.0. In *ICSE*, pages 741–742, 2004.

[18] J. Vera, L. Perrochon, and D. C. Luckham. Event-based execution architectures for dynamic software systems. In *WICSA*, pages 303–318, 1999.