A Logical Interface Description Language for Components

F. Arbab[1], F.S. de Boer[2], and M.M. Bonsangue[1]

[1] CWI, Amsterdam, The Netherlands
{farhad,marcello}@cwi.nl

[2] Utrecht University, The Netherlands
frankb@cs.uu.nl

Abstract. Motivated by our earlier work on the IWIM model and the Manifold language, in this paper, we attend to some of the basic issues in component-based software. We present a formal model for such systems, a formal-logic-based component interface description language that conveys the observable semantics of components, a formal system for deriving the semantics of a composite system out of the semantics of its constituent components, and the conditions under which this derivation system is sound and complete. Our main results in this paper are the theorems that formulate the notion of compositionality and the completeness of the derivation system that supports this property in a component-based system.

1 Introduction

Building applications out of software components is currently a major challenge for Software Engineering. The urgency and importance of this challenge are intensified by the continuous rapid growth of the supply and demand for software (components) on the internet, and the prospect of mobile computing. There are close ties between many of the issues investigated in the coordination research community in the past decade or so, on the one hand, and some of the basic problems in Component Based Software Engineering, on the other.

Motivated by our earlier work on the IWIM model and the Manifold language, in this paper we introduce a formal logic-based interface description language for components in component-based systems. We consider components as black box computational entities that communicate asynchronously via unbounded FIFO buffers. Each such FIFO buffer is called a channel and has a system-wide unique identity. The identity of a channel can also be communicated as a value through channels. This allows dynamic reconfiguration of channel connections among the components of a system.

The interface of a component describes its observable behavior abstracting away its implementation in a particular programming language. The interface of a component contains five elements: a name, a channel signature, and three predicates, namely a blocking invariant, a precondition, and a postcondition.

The name of a component uniquely identifies the component within a system. The channel signature of a component is a list of channels representing its initial connections. The blocking invariant is a predicate that specifies the possible deadlock behavior of the component. The precondition is a predicate that specifies the contents of the buffers of the initial external channels (i.e., the ones in the channel signature) of the component. The postcondition is a predicate that specifies the contents of the buffers of the external channels that exist upon termination.

In order to simplify our presentation in this paper, we restrict ourselves to component-based systems that consist of a static number of components and channels, although the connections in the system can change dynamically and in an arbitrary manner. Semantically, we describe the behavior of a component by a transition system, abstracting away from its internal details and the language of its implementation. We define the observable behavior of a component in terms of sequences of values, one for each channel-end that the component has been connected to. Thus, we abstract away the ordering among the communications on different channels. The observable behavior of a component-based system is given by the set of final global states of successfully terminating computations, provided that the system is deadlock-free. The existence of a deadlocking computation is considered a fatal error. A global state records for each channel the contents of its buffer.

The main contribution of this paper is to show that it is possible to reason about the correctness of an entire system compositionally in terms of the interface specifications of its components, abstracting away their internal implementation details. Our notion of correctness of a component-based system is based on the above-mentioned concept of observable behavior. This extends the usual notion of partial correctness by excluding deadlocks.

Compositionality is a highly desirable, but elusive, property for formal models of component-based systems. For compositionality to hold, the formal system that relates the semantics of the whole system to that of its individual components must constitute a proof method that is both sound and complete. We show that our proof method is generally sound. On the other hand, it is not generally possible to derive the formal semantics of a whole system as a composition of the local semantics of its components only. Consequently, completeness of our proof method does not generally hold. However, we show that it is possible to obtain completeness for component-based systems that satisfy certain restrictions. Indeed, we show that these restrictions are both necessary and sufficient conditions for completeness.

To achieve completeness, we impose two restrictions on component-based systems. First, we restrict to channels that are one-to-one and uni-directional. This means that every channel is an exclusively point-to-point communication medium between a single producer and a single consumer. The producer or the consumer of a channel loses its exclusive control of its channel-end by writing its identifier end to another channel. Subsequently, a component may dynamically (re)gain the exclusive control of a specific end of a channel, simply by reading its identifier as a value from another channel. This allows dynamic reconfiguration of channel connections among the components in a system.

The second restriction imposes certain constraints on the forms of global non-determinism allowed in a system. We elaborate on this in Section 5.1.

We proceed as follows. In the next section we define a semantic model for components and define its observable behavior. In Section 3, we define the semantics of a component-based system. In Section 4, we introduce a formal language to describe interfaces of components, and formally define its semantics. Finally, in Section 5, we introduce a sound compositional proof system that allows to derive a system-wide correctness specification from the interface specifications of its components. We end this section by showing the completeness of the proof system for a certain class of component-based systems.

1.1 Comparison with Related Work

Over the past few years several component infrastructure technologies, such as Corba [16], ActiveX [14], and JavaBeans [12], have been developed, each of which embodies a different notion of "software components". Indeed, none of these technologies offers a formal definition of a component, and none of the twenty-or-so informal definitions for "component" commonly found in the literature on component-based systems is exact enough to be formalizable. Following [4], we strongly advocate a formal framework for componentware, to reflect the essential concepts in existing component-based approaches.

Our model for component-based systems is inspired by works in (1) architectural description languages, like UniCon [18], and (2) coordination languages, like Manifold [3]. Our model supports heterogeneity and reusability of components and provides modularity of description. Components communicate asynchronously and anonymously via identifiable channels. Thus, our model differs from models of asynchronously communicating process like CSP [13], parallel object-oriented languages [6], and actor languages [1], where communication between the processes, objects, or actors, is established via their identities.

Our notion of the interface of a component includes a description of its observable behavior. This is in contrast to most current interface description languages [16] which specify only some syntactic characteristics of a component, and thus reduce the analysis of a component-based system to mere syntactic consistency checks.

To the best of our knowledge, only [5] takes a similar semantical approach to the definition of a component interface. However, their model does not allow for dynamic reconfiguration of the connections, and gives no formal language for the description of the semantical information in a component interface. They, as well as many other systems, allow multiple interfaces for a single component. While our model has no specific features to support multi-interface components, it does not preclude them either: our model simply deals with component interfaces and is oblivious to the possible associations of (one or more) component interfaces with actual components.

Our semantic approach is based on the one taken in [7] for a language, introduced in [2], for describing confluent dynamically reconfigurable data-flow networks. In this paper we abstract away the syntactic representation of components, show the necessity of confluence to obtain a compositional semantics, and present a proof method for reasoning about the correctness of a component-based system. Generalization of data-flow networks for describing dynamically reconfigurable or mobile networks has also been studied in [9] and [11] for a different notion of observables using the model of stream functions.

Our computational model provides a framework for the study of the semantic basis of assertional proof methods for communicating and mobile processes. As such, our approach is different than the various process algebras for mobility, like the π-calculus [15] or the ambient calculus [10].

2 The Observable Behavior of Components

Components are the basic entities of a system. They interact by means of exchanging values via channels. A channel is an unbounded FIFO buffer. It represents a reliable and directed flow of information from its source to its sink. A component may send a value to a channel only if it is connected to its source. Similarly, it may receive a value from a channel only if it is connected to its sink. The identity of the source or the sink of a channel itself can also be communicated via a channel. As such, the connection topology in a system can dynamically change. Initially, we assume that each component is connected to a given set of sources and/or sinks of some channels. This defines the initial connection topology of a system.

In this section we introduce a formal model of the observable behavior of a component in terms of a transition system that abstracts away its internal behavior [8]. The internal behavior itself may be implemented in different programming languages.

For the rest of this paper, let Chan be a set of channel identities with typical elements $c, c', \ldots$, and let $\overline{\mathit{Chan}}$ be the set $\{\overline{c} \mid c \in \mathit{Chan}\}$. For a channel $c \in \mathit{Chan}$, we denote by $\overline{c}$ its source-end and associate the channel identity c with its sink-end. The source-end $\overline{c}$ and the sink-end c of a channel c are also called its channel-ends. Furthermore, let Val be a set of values, with typical elements $u, v, \ldots$, that includes both c and $\overline{c}$ for every $c \in \mathit{Chan}$. We denote by Act the set of communication actions of the forms $\overline{c}!v$ and $c?v$ for each $c \in \mathit{Chan}$ and $v \in \mathit{Val}$, which respectively denote the sending and the receiving of a value v through a channel c. We assume the read action $c?v$ is destructive: as a result of this action, a value v is irrevocably removed from the FIFO buffer of c.

Definition 1. *A component C is specified by a transition system $\langle L, i, r, \longrightarrow \rangle$, where L is a set of (control) locations, with typical element l; $i \in L$ is an initial location; r is a set of channel-ends to which the component is initially connected; and $\longrightarrow \subseteq L \times \mathit{Act} \times L$ is a transition relation that describes the communication behavior of the component. As usual, we use $s \xrightarrow{a} s'$ to indicate that $(s, a, s') \in \longrightarrow$.*

Intuitively, a component may send a value v to the source-end of a channel c by performing a $\overline{c}!v$ action, and may receive a value v from the sink-end of a channel c by performing a $c?v$ action. If in some location, a component is willing to receive a value from the sink of a channel c, then it should be willing to accept any value from that sink. We call this property '*input reactiveness*' formally expressed as the following condition that must be satisfied by the transition relation of every component $\langle L, i, r, \longrightarrow \rangle$:

$$\forall l \xrightarrow{c?v} l' \,.\, \forall u \in \mathit{Val} \,.\, \exists l'' \in L \,.\, l \xrightarrow{c?u} l'' \,.$$

In other words, we restrict to component-based systems that cannot put selection constraints on the values they receive. This restriction is introduced for technical convenience only (specifically, it allows a slightly simpler deadlock analysis).

A component can communicate only via channel-ends to which it is actually connected. Initially, a component $C \widehat{=} \langle L, i, r, \longrightarrow \rangle$ is connected only to the channel-ends in its r. Other channel-ends can be used only after the component receives their identities through other channels. Formally, we require that, for every computation $i \xrightarrow{a_1} l_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} l_n$ of C:

1. if $a_n = c?v$ then $c \in r$ or there exists a preceding input $a_j = d?c$, $1 \leq j < n$ and $d \neq c$; and
2. if $a_n = \overline{c}!v$ then $\overline{c} \in r$ or there exists a preceding input $a_j = d?\overline{c}$, $1 \leq j < n$.

Note that the source end of a channel may be received through the sink end of the same channel, whereas its sink end, of course, cannot be received through the channel itself.

Next, we define the observable behavior of a component by mapping each computation to the sequence of values the component sends or receives through each channel-end. Information about the deadlock behavior of a component will be given in terms of a so-called ready set [17] consisting of those channel-ends on which the component is waiting for input. Note that as such, we do not have any information about the ordering among the communications of a component through different channel-ends. In practice, this abstraction simplifies reasoning about the correctness of the entire systems, as the value sent or received by a component at a particular point in time will be independent of the time other values are sent or received through other channels. However, we will record some information about when channels are exported.

To record when a channel (end) is exported by a component, we extend the set of values with a special element $\gamma \notin \mathit{Val}$. Let $\mathit{Val}_\gamma = \mathit{Val} \cup \{\gamma\}$. We define a component state to be a function s that maps the sink of each channel $c \in \mathit{Chan}$ to a sequence $s(c) \in \mathit{Val}_\gamma^*$ of values received from the channel c, and the source of each channel $c \in \mathit{Chan}$ to a sequence $s(\overline{c}) \in \mathit{Val}_\gamma^*$ of values sent to the channel c. The occurrence of the symbol γ in a sequence $w_1 \cdot \gamma \cdot w_2 \cdot \gamma \cdots$ indicates that the channel-end c (or $\overline{c}$) was exported in between the sequences of read (or sent) values $w_1, w_2, \ldots$.

For each component $C \widehat{=} \langle L, i, r, \longrightarrow \rangle$ we formally model its observable behavior by means of a transition relation $\longrightarrow\!\!\!\!\!\rightarrow$ on configurations of the form $\langle l, s \rangle$, where s denotes a component state as defined above. This relation is defined as the least relation which satisfies the following rules:

$$\frac{l \xrightarrow{c?v} l'}{\langle l, s\rangle \longrightarrow\!\!\!\!\!\rightarrow \langle l', s[s(c)\cdot v/c]\rangle}$$

$$\frac{l \xrightarrow{\overline{c}!v} l' \text{ and } v \notin \mathit{Chan} \cup \overline{\mathit{Chan}}}{\langle l, s\rangle \longrightarrow\!\!\!\!\!\rightarrow \langle l', s[s(\overline{c})\cdot v/\overline{c}]\rangle}$$

$$\frac{l \xrightarrow{\overline{c}!v} l' \text{ and } v \in \mathit{Chan} \cup \overline{\mathit{Chan}}}{\langle l, s\rangle \longrightarrow\!\!\!\!\!\rightarrow \langle l', s[s(\overline{c})\cdot v/\overline{c}][s(v)\cdot\gamma/v]\rangle}$$

Here $s[a/x]$ denotes the function that maps x to a and otherwise acts as s. The effect of an input communication on a state s is that the received value is appended to the sequence $s(c)$ of values received so far from the channel c. Similarly, the effect of an output on a state s is that the sent value is appended to the sequence $s(\overline{c})$ of values sent so far along the channel c. Moreover, if the sent value is a channel-end, γ is appended to the sequence associated with that channel-end.

We now formally define the observable behavior of a component.

Definition 2. *Let $C \widehat{=} \langle L, i, r, \longrightarrow\rangle$ be a component and s be a component state. We denote by $\longrightarrow\!\!\!\!\!\rightarrow^*$ the reflexive transitive closure of the transition relation $\longrightarrow\!\!\!\!\!\rightarrow$. Moreover, $\langle l, s\rangle \not\longrightarrow\!\!\!\!\!\rightarrow$ indicates that no further transition is possible from $\langle l, s\rangle$, i.e., l is a final location. Finally, let D be the set of locations l from which only input transitions $l \xrightarrow{c?v} l'$ are possible. The observable behavior $\mathcal{O}(C)(s)$ of component C in an initial state s is defined as a pair $\langle T, R\rangle$ where:*

$$T = \{s' \mid \langle i, s\rangle \longrightarrow\!\!\!\!\!\rightarrow^* \langle l, s'\rangle \not\longrightarrow\!\!\!\!\!\rightarrow\}$$
$$R = \{(s', \{c \mid l \xrightarrow{c?v} l'\}) \mid \langle i, s\rangle \longrightarrow\!\!\!\!\!\rightarrow^* \langle l, s'\rangle,\ l \in D\}\,.$$

Thus, the semantics of a component in isolation consists of the set T of all final states of successfully terminating computations, plus the set R of all those reachable states that may give rise to a deadlock, together with a corresponding ready-set. Given a reachable state which may give rise to a deadlock, its corresponding *ready-set* contains all channels on which the component is ready to perform an input action.

3 Component-Based Systems

A component-based system π consists of a finite collection of components $C_1 \parallel \cdots \parallel C_n$. In order to specify the dynamics of a system, we introduce the set Σ of system states, with typical element σ. A system state σ is a function that maps each channel sink c and channel source $\overline{c}$ to a sequence of indexed values (k, v), where $k \in \{1, \ldots, n\}$ and $v \in \mathit{Val}_\gamma$. The index k indicates that it was the component C_k that sent or received the value v through the given channel end. We restrict ourselves to those system states σ that are *prefix invariant*, i.e., for every channel the sequence of values received from its source is a prefix of the sequence of values delivered through its sink:

$$\forall c \in \mathit{Chan} : \mathit{Val}(\sigma(\overline{c})) \sqsubseteq \mathit{Val}(\sigma(c)).$$

Here, $\sqsubseteq$ is the prefix relation among sequences, and, for a sequence w of indexed values $(k_1, v_1), (k_2, v_2), \cdots (k_n, v_n)$, $\mathit{Val}(w)$ denotes the sequence of values $v_1, v_2, \cdots v_n$ obtained from w by removing the indices and the occurrences of the control symbol γ, if any.

Observe that, given a system state σ and a channel c, the sequence that is the difference between $\mathit{Val}(\sigma(c))$ and $\mathit{Val}(\sigma(\overline{c}))$ is the contents of the buffer of c in σ, denoted as $\mathit{buf}(\sigma, c)$.

Let $C_k \widehat{=} \langle L_k, i_k, r_k, \longrightarrow_k \rangle$, $k = 1, \ldots, n$, and $\sigma \in \Sigma$. The observable behavior of a component-based system $\pi = C_1 \parallel \cdots \parallel C_n$ is defined in terms of a global transition relation $\longrightarrow$ on global configurations of the form $\langle l, \sigma \rangle$, where $l \in L_1 \times \cdots \times L_n$ and σ denotes a system state as defined above. We define $\longrightarrow$ as the least transition relation satisfying the following three rules:

$$\frac{l_k \xrightarrow{c?v}_k l'_k \text{ and } \mathit{buf}(\sigma, c) = w \cdot v}{\langle (l_1, \ldots, l_k, \ldots, l_n), \sigma \rangle \longrightarrow \langle (l_1, \ldots, l'_k, \ldots, l_n), \sigma[\sigma(c) \cdot (k, v)/c] \rangle}$$

$$\frac{l_k \xrightarrow{\overline{c}!v}_k l'_k \text{ and } v \notin \mathit{Chan} \cup \overline{\mathit{Chan}}}{\langle (l_1, \ldots, l_k, \ldots, l_n), \sigma \rangle \longrightarrow \langle (l_1, \ldots, l'_k, \ldots, l_n), \sigma[\sigma(\overline{c}) \cdot (k, v)/\overline{c}] \rangle}$$

$$\frac{l_k \xrightarrow{\overline{c}!v}_k l'_k \text{ and } v \in \mathit{Chan} \cup \overline{\mathit{Chan}}}{\langle (l_1, \ldots, l_k, \ldots, l_n), \sigma \rangle \longrightarrow \langle (l_1, \ldots, l'_k, \ldots, l_n), \sigma[\sigma(\overline{c}) \cdot (k, v)/\overline{c}][\sigma(v) \cdot (k, \gamma)/v] \rangle}$$

A component receives a value v from a channel-end c only if v is the first element of the buffer of channel c (which is thus non-empty). Otherwise it blocks. On the other hand, a component can always append a value to the buffer of a channel c by sending it through the channel-end $\overline{c}$.

Let $\overline{i}$ denotes the tuple $(i_1, \ldots, i_n)$ of initial locations. By $\overline{l}$ we denote a tuple $(l_1, \ldots, l_n)$ of locations l_i, $i = 1, \ldots, n$. Furthermore, we use the symbol $\delta \notin \Sigma$ in $\langle \overline{i}, \sigma \rangle \Rightarrow \delta$ to denote the existence of a *deadlocking computation* starting from state σ. This means that $\langle \overline{i}, \sigma \rangle \longrightarrow^* \langle \overline{l}, \sigma' \rangle$ and from the configuration $\langle \overline{l}, \sigma' \rangle$ no further transition is possible, although for some location l_k of $\overline{l}$ there exists a transition $l_k \xrightarrow{a}_k l'_k$, for some communication action a and location l'_k. Furthermore, $\langle \overline{i}, \sigma \rangle \Rightarrow \sigma'$ indicates a computation that starts from σ and *successfully terminates* in a system state σ'. This means that $\langle \overline{i}, \sigma \rangle \longrightarrow^* \langle \overline{l}, \sigma' \rangle$ and for all locations l_k of $\overline{l}$, communication action a, and location l'_k, there is no transition $l_k \xrightarrow{a}_k l'_k$.

We now define the observable semantics of a component-based system.

Definition 3. *Let $\pi = C_1 \parallel \cdots \parallel C_n$, with $C_k \widehat{=} \langle L_k, i_k, r_k, \longrightarrow_k \rangle$, $k = 1, \ldots, n$ and let $\overline{i}$ denote the tuple $(i_1, \ldots, i_n)$ of initial locations. We define*

$$\mathcal{O}(\pi)(\sigma) = \begin{cases} \delta & \text{if } \langle \overline{i}, \sigma \rangle \Rightarrow \delta, \\ \{\sigma' \mid \langle \overline{i}, \sigma \rangle \Rightarrow \sigma'\} & \text{otherwise.} \end{cases}$$

Thus, $\mathcal{O}(\pi)(\sigma)$ collects all the final states of the system that correspond to its successfully terminating computations, if it involves no deadlocks. Deadlock itself is considered a fatal error.

4 A Logical Interface Description Language

We introduce a formal assertion language for specifying the observable behavior of a component $C \widehat{=} \langle L, i, r, \longrightarrow \rangle$ via an interface. The interface of a component consists of the following:

- the name of the component;
- an initial set of external connections (the sinks and/or sources of some channels);
- a blocking invariant;
- a precondition;
- a postcondition.

The blocking invariant specifies the possible deadlock behavior of a component. The precondition specifies the contents of the buffers of the initial external channels, and the postcondition specifies the sequences of values received from and delivered to the external channels that exist upon termination.

The above specification of a component involves a multi-sorted assertion language which includes the sort *Chan* of channel-sinks and the sort $\overline{Chan}$ of channel-sources. In fact, c and $\overline{c}$ are introduced as constants in the assertion language for every $c \in Chan$. Apart from the sort of values that can be transmitted along channels (which thus includes the set $Chan \cup \overline{Chan}$) our specification language includes the sort of (finite) sequences of values. Finally, we assume that the sort set of *Chan* and $\overline{Chan}$ is given.

We denote by *Var*, with typical elements $x, y, z, \ldots$, the set of all variables. For each sort we assume that a set of variables of that sort is given, and that these sets of variables are disjoint for different sorts. We denote by $\mathcal{S}$ the underlying signature of many-sorted operators f and predicates p. It includes, for example, the usual sequence operations like append, prefix, etc., and the usual set operations of union, intersection, etc. An example of a useful operator is $\bar{\cdot}$, that can be applied to a channel-end resulting into the other channel-end. Thus applying this operator to the sink-end of a channel c returns its corresponding source-end $\overline{c}$, and, likewise, $\overline{\overline{c}} = c$.

Given the above multi-sorted signature $\mathcal{S}$, we introduce the following set of expressions of the assertion language (we omitt sort restrictions).

Definition 4. *An expression e of the assertion language is defined as follows (we omit the type information).*

$$e ::= c \mid \overline{c} \mid x \mid e\!\downarrow \mid e\!\downarrow_k \mid f(e_1, \ldots, e_n),$$

where $k \in \{1, \ldots, n\}$, $f \in \mathcal{S}$ denotes an operator, $c \in Chan$, and $x \in Var$.

The *local* semantics of an expression e is formally given by $\mathcal{E}(e)(\omega)(s)$, where ω is a function that assigns to each variable a corresponding value (of the correct type), and s is component state. We have that

- $\mathcal{E}(c)(\omega)(s) = c$;
- $\mathcal{E}(\overline{c})(\omega)(s) = \overline{c}$;
- $\mathcal{E}(x)(\omega)(s) = \omega(x)$;
- $\mathcal{E}(e\downarrow)(\omega)(s) = s(\mathcal{E}(e)(\omega)(s))$
- $\mathcal{E}(e\downarrow_k)(\omega)(s) = s(\mathcal{E}(e)(\omega)(s))$
- $\mathcal{E}(f(e_1, \ldots, e_n)(\omega)(s) = f(\mathcal{E}(e_1)(\omega)(s), \ldots, \mathcal{E}(e_n)(\omega)(s))$, associating an operator f with its interpretation.

The constants c and $\overline{c}$, thus, denote the sink and the source of the channel c, respectively. The value of a variable is given by ω. Given an expression e denoting a channel-end, the expressions $e\downarrow$ and $e\downarrow_k$ both denote the sequence of values associated with that channel-end in the component state s. We will see later that these two expressions will receive a different interpretation at a global level. The definition of the semantics of a complex expression is standard.

Next, we introduce the syntax of assertions.

Definition 5. *An assertion ϕ of the assertion language is defined as follows.*

$$\phi ::= p(e_1, \ldots, e_n) \mid \neg\phi \mid \phi \wedge \phi \mid \exists x(\phi)$$

Here $p \in \mathcal{S}$ denotes a many-sorted predicate, and $x \in Var$ is a variable.

By $s, \omega \models \phi$ we denote that the assertion ϕ is true with respect to a variable-assignment ω and a component state s. This definition is standard. For example,

$$s, \omega \models p(e_1, \ldots, e_n) \text{ if and only if } p(\mathcal{E}(e_1)(\omega)(s), \ldots, \mathcal{E}(e_n)(\omega)(s)),$$

associates a predicate p with its interpretation. Thus, given the prefix relation $\sqsubseteq \in \mathcal{S}$ on sequences, the assertion

$$c\downarrow \sqsubseteq \overline{d}\downarrow$$

expresses that the sequence of values received through the channel c is sent along the channel d.

As another example, we show how to express in our assertion language that a channel x has been known to a given component initially connected to channels in a set r. In order to do so, we assume the presence of an operator *setchan* in our signature $\mathcal{S}$ whose interpretation is to return the set of sinks and sources of all channels occurring in a given sequence of values. Moreover, we use $\mu y(\phi)$ as a shorthand for a set-quantifier that gives the smallest set y for which ϕ holds (it is not hard to see that such a quantifier can be expressed in our assertion language). That a channel x has been known to a component can now be expressed as

$$x \in \mu y\ (r \subseteq y \wedge \forall z(z \in y \rightarrow setchan(z\downarrow) \subseteq y)).$$

In other words, the set of channels that have been known to a component is the smallest set containing the channels to which the component is initially connected, plus, for every channel, those channels stored in its associated sequence of values.

It is worthwhile to observe that we have the following algebraic characterization of the operator *setchan*:

$$\begin{aligned}
&setchan(\varepsilon) &&= \emptyset \\
&setchan(c \cdot w) &&= \{c\} \cup setchan(w) \\
&setchan(\overline{c} \cdot w) &&= \{\overline{c}\} \cup setchan(w) \\
&setchan(v \cdot w) &&= setchan(w) \quad v \notin Chan \cup \overline{Chan}.
\end{aligned}$$

Here ε denotes the empty sequence. Generally, reasoning about the properties of channels as expressed by our assertion language involves algebraic axiomatizations of the data types of sets and sequences.

The following definition introduces the notion of the interface of a component.

Definition 6. *The observable interface of a component is a labeled tuple of the form*

$$\langle Id\colon C, Chan\colon r, Inv\colon I(z), Pre\colon \phi(r), Post\colon \psi(r)\rangle$$

Here z is a variable that ranges over sets of channel sinks only, and $\phi(r)$ and $\psi(r)$ denote assertions with occurrences of r.

The blocking invariant $I(z)$, which denotes an assertion with free occurrences only of the variable z, specifies the possible deadlock behavior of C. It holds in all those component states where the component C is committed to get a value from channels in z, possibly blocking it. The assertions $\phi(r)$ and $\psi(r)$ denote the usual pre- and postconditions, where r denotes the set of channel-ends to which C is initially connected. For notational convenience only, we assume that initially the buffers of all the channels in r are empty (so we do not need a precondition). We then abbreviate a component interface by a triple $I(z)\colon C\{\psi(r)\}$. As a simple example, the interface

$$z = \{c\} \vee z = \{d\}\colon C\{c{\downarrow}= \overline{d}{\downarrow}\}$$

denotes a component named C, initially connected to the sinks of two channels, namely c and d and to the source of d. The component receives values from either c or d, and upon termination every values it has read from c it has been output to d in the same order.

Formally, we have the following semantics for component interfaces:

Definition 7. *Let $C \widehat{=} \langle L, i, r, \longrightarrow\rangle$ be a component, and let $\mathcal{O}(C) = \langle T, R\rangle$ be its observable semantics. We define*

$$\models I(z)\colon C\{\psi(r)\}$$

if for all variable assignment ω, component state $s \in T$ and ready pair $(s', r') \in R$, we have $s, \omega \models \psi(r)$ and $s', \omega \models I(r')$. Here $I(r')$ denotes the result of replacing every occurrence of z in I by r'.

Basically, we have the same assertion language for the specification of correctness properties at the level of a system of components.

Definition 8. *Let $\pi = C_1 \parallel \cdots \parallel C_n$ be a component-based system, with $r_1, \ldots, r_n$ sets of initial connections for each component in the system. A system correctness specification for π is of the form $\{\phi(r)\}\pi\{\psi(r)\}$, where $r = r_1 \cup \cdots \cup r_n$, and $\phi(r)$ and $\psi(r)$ denote assertions with occurrences of r.*

The assertions $\phi(r)$ and $\psi(r)$ denote the usual pre- and postconditions. For technical convenience only, we assume that the buffers of all channels in r are empty. Consequently, we do not need to consider the precondition. Thus, we abbreviate a system specification as $\pi\{\psi(r)\}$.

In order to define the semantics of a system-wide correctness specification for a system $\pi = C_1 \parallel \cdots \parallel C_n$, we introduce a different *system-wide* interpretation for the assertion language. The semantics of an expression e is now given by $\mathcal{G}(e)(\omega)(\sigma)$, where σ is a system state of π. The main difference between the system-wide and the component-level interpretations is that we define for expression e of sort channel-source or channel-sink,

- $\mathcal{G}(e\downarrow)(\omega)(\sigma) = \sigma(\mathcal{G}(e)(\omega)(\sigma))$
- $\mathcal{G}(e\downarrow_k)(\omega)(\sigma) = \sigma(\mathcal{G}(e)(\omega)(\sigma))\downarrow_k$,

where $\downarrow_k$ projects a sequences of indexed values into the sequence of values (including the control symbol γ) indexed by k. Algebraically,

$$\begin{aligned} \varepsilon\downarrow_k &= \varepsilon \\ ((k,v)\cdot w)\downarrow_k &= v\cdot(w\downarrow_k),\ v \in \mathit{Val}_\gamma \\ ((j,v)\cdot w)\downarrow_k &= w\downarrow_k,\ j \neq k. \end{aligned}$$

At the level of the global assertion language, the expressions $c\downarrow$ and $\overline{c}\downarrow$, thus, denote sequences of *indexed* values (k, v), where $k \in \{1, \ldots, n\}$ is the index for the actor component C_k involved in the reading or the writing of v.

Analogous to the component-level interpretation, we denote by $\sigma, \omega \models \phi$ that ϕ is true with respect to the variable assignment ω and system state σ. An assertion ϕ is valid, indicated by $\models \phi$, if $\sigma, \omega \models \phi$, for every σ and ω.

As an example, we have for every channel c, the global validity of the assertion axiomatizing the FIFO nature of c:

$$\mathit{Val}(c\downarrow) \sqsubseteq \mathit{Val}(\overline{c}\downarrow),$$

where Val is algebraically characterized by

$$\begin{aligned} \mathit{Val}(\varepsilon) &= \varepsilon \\ \mathit{Val}((k,\gamma)\cdot w) &= \mathit{Val}(w) \\ \mathit{Val}((k,v)\cdot w) &= v\cdot \mathit{Val}(w),\ v \in \mathit{Val}. \end{aligned}$$

The global validity of the above assertion follows from the fact that all system states are prefix invariant.

As another example, given two sequences w_1 and w_2 and that $w_1 - w_2$ yields the suffix of the sequence w_1 determined by its prefix w_2, the global interpretation of the expression

$$Val(\overline{c}\downarrow) - Val(c\downarrow)$$

denotes the contents of the buffer of a channel c. In the sequel, we abbreviate this expression as $buf(c)$. Note that, generally, we cannot denote the buffer of a channel by an expression interpreted in the state of a component.

We formally define the semantics of a system-wide correctness specification in terms of the above system-wide interpretation of the assertion language.

Definition 9. *Let $C_1, \ldots, C_n$ be some components with (disjoint) initial connection sets $r_1, \ldots r_n$, respectively. Let $\pi = C_1 \parallel \cdots \parallel C_n$ be a component-based system, and $r = r_1 \cup \cdots \cup r_n$. We define $\models \pi\{\psi(r)\}$ if $\mathcal{O}(\pi)(\sigma_0) \neq \delta$ and for all $\sigma \in \mathcal{O}(\pi)(\sigma_0)$ we have that $\sigma, \omega \models \psi(r)$, for every variable assignment ω. Here (for simplicity) σ_0 is the system state mapping each channel in r to the empty sequence ε.*

A global specification $\pi\{\psi(r)\}$, thus, is valid if π does not have a deadlocking computation and every successfully terminating computation in π results in a system state that satisfies $\psi(r)$.

4.1 Expressing Absence of Deadlocks

As a major example, we show how to express the absence of deadlocks in a system $\pi = C_1 \parallel \cdots \parallel C_n$ in our assertion language. First we need to introduce the assertion $\phi\downarrow_k$ that we derive from ϕ by replacing every occurrence of the operator $\downarrow$ by $\downarrow_k$. As discussed previously, this latter operator selects from a labeled sequence of values the sequence of only those values labeled by the index k.

Assume the interface specifications $I_1(z_1){:}C_1\{\psi_1\}, \ldots, I_n(z_n){:}C_n\{\psi_n\}$ are given for the components of π. Let $\overline{I}$ and $\overline{\psi}$ denote the sequences of assertions $I_1, \ldots, I_n$ and $\psi_1, \ldots, \psi_n$, respectively. Under our system-wide interpretation, the following assertion then defines $\delta(\overline{I}, \overline{\psi})$ to holds on all possible deadlock states in the system π.

$$\delta(\overline{I}, \overline{\psi}) \hat{=} \bigwedge_i (I'_i \vee \psi'_i) \wedge \bigvee_i I'_i \wedge \bigwedge_i (I'_i \rightarrow \forall x \in z_i(buf(x) = \epsilon)).$$

Here, I'_i denotes the assertion $I_i\downarrow_i$ and, similarly, ψ'_i denotes the assertion $\psi_i\downarrow_i$. Note that the logical structure of this assertion reflects the semantic definition of a global deadlock situation given that I'_i represents the state of the component C_i as it tries to input from a channel in the set z_i, and ψ'_i represents its state of the component C_i upon termination: the first conjunct expresses that each component C_i of the system is either terminating in a state satisfying ψ'_i or it tries to input from a channel in the set z_i in a state satisfying I'_i. The second conjunct guarantees that there esists at least one component that tries to input from a channel, and the third conjunct expresses that all those components are actually blocked beacause the channels on which they are inputting are empty.

We explain the above assertion using a simple system $\pi = C_1 \parallel C_2$. In π, the component C_1 repeatedly writes a value to the channel d and subsequently reads a value from the channel c. The component C_2, on the other hand, repeatedly reads a value from d and subsequently writes that value to c. Let $r_1 = \{c, \overline{d}\}$ and $r_2 = \{\overline{c}, d\}$ be the sets of initial connections of C_1 and C_2, respectively. Also, let $I_1(z_1)$ and $I_2(z_2)$ denote the assertions

$$|c{\downarrow}| < |\overline{d}{\downarrow}| \wedge z_1 = \{c\} \wedge \forall z \notin r_1(z{\downarrow} = \epsilon)$$

and

$$|d{\downarrow}| = |\overline{c}{\downarrow}| \wedge z_2 = \{d\} \wedge \forall z \notin r_2(z{\downarrow} = \epsilon),$$

respectively, where the operation $|w|$ gives the length of the sequence w. Intuitively, the assertion $I_1(z_1)$ states that the number of values read from the channel c by the component C_1 is strictly smaller than the number of values written to the channel d by C_1 as it is about to read from c. Furthermore, it states that the component C_1 reads only from the channel c and writes only to the channel d. On the other hand, the assertion $I_2(z_2)$ states that the number of values read from d by the component C_2 is exactly equal to the number of values read from c by C_2, as it is about to read from d. Furthermore, it states that the component C_2 reads only from the channel d and writes only to the channel c.

We assume that C_1 and C_2 do not terminate, so that we can take *false* as the postcondition for both components. The assertion $\delta(\overline{I}, \overline{\psi})$ for $\overline{I} = I_1(z_1), I_2(z_2)$ and $\overline{\psi} = \mathit{false}, \mathit{false}$, thus, logically reduces to the assertion

$$I_1{\downarrow_1} \wedge I_2{\downarrow_2} \wedge \mathit{buf}(c) = \epsilon \wedge \mathit{buf}(d) = \epsilon, \tag{1}$$

which holds if the system can deadlock. Next, we prove that this assertion leads to a contradiction. We have

$$|c{\downarrow_1}| < |\overline{d}{\downarrow_1}| \wedge |d{\downarrow_2}| = |\overline{c}{\downarrow_2}|. \tag{2}$$

Moreover, from $\mathit{buf}(c) = \epsilon \wedge \mathit{buf}(d) = \epsilon$ it follows that

$$|c{\downarrow}| = |\overline{c}{\downarrow}| \wedge |d{\downarrow}| = |\overline{d}{\downarrow}|. \tag{3}$$

Since C_1 and C_2 are the only components and

$$\forall z \notin r_1(z{\downarrow_1} = \epsilon) \text{ and } \forall z \notin r_2(z{\downarrow_2} = \epsilon)$$

we derive from (3) the assertion

$$|c{\downarrow_1}| = |\overline{c}{\downarrow_2}| \wedge |d{\downarrow_2}| = |\overline{d}{\downarrow_1}|, \tag{4}$$

that is in contradiction with assertion (2). Since the assertion (1), above, describes all possible deadlock situations, we conclude that the system π cannot deadlock.

5 Composing Component Interfaces

In this section, we introduce a compositional proof system that allows us to derive a system-wide correctness specification from the interface specifications of the constituent components of a system.

In order to formulate this proof system, we observe that the following property holds for this projection operator.

Lemma 1. *For every assertion ϕ, variable assignment ω, and system state σ we have that*

$$\sigma, \omega \models \phi{\downarrow_k} \text{ if and only if } \sigma{\downarrow_k}, \omega \models \phi\,,$$

where $\sigma{\downarrow_k}$ denotes the component state resulting from applying ${\downarrow_k}$ to every sequence of labeled values $\sigma(c)$ and $\sigma(\overline{c})$ for all $c \in Chan$, i.e., $\sigma{\downarrow_k}(c) = \sigma(c){\downarrow_k}$.

In other words, the above lemma states that the system-wide interpretation of $\phi{\downarrow_k}$ is the same as the component-level interpretation of ϕ.

We now formulate our proof system for deriving system-wide correctness formulas.

Definition 10. *Let $\pi = C_1 \parallel \cdots \parallel C_n$ be a component based system and let $I_1(z_1){:}C_1\{\psi_1\}$, ..., $I_n(z_n){:}C_n\{\psi_n\}$ be the interfaces of its components. We denote by $\vdash \pi\{\psi\}$ that the system correctness formula $\pi\{\psi\}$ is derivable from the following proof system:*

$$\frac{I_i(z_i){:}\ C_i\{\psi_i\} \text{ and } \models \neg\delta(\overline{I};\overline{\psi})}{\pi\{\bigwedge_i(\psi_i{\downarrow_i})\}} \qquad \frac{\pi\{\psi\} \text{ and } \models \psi \to \psi'}{\pi\{\psi'\}}$$

where $\overline{I} = I_1(z_1), \ldots, I_n(z_n)$ and $\overline{\psi} = \psi_1, \ldots, \psi_n$.

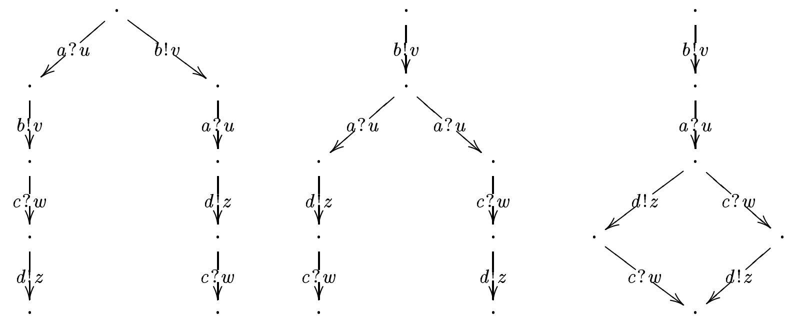
In order to prove the soundness of the first and main rule of our proof system, we first need to show that the validity of the assertion $\neg\delta(\overline{I}, \overline{\psi})$ guarantees the absence of deadlocks. Indeed, the validity of the component-level correctness specifications $I_i(z_i){:}\ C_i\{\psi_i\}$, for each $i \in \{1, \ldots, n\}$, implies that every deadlocked system state σ of π satisfies the assertion $\delta(\overline{I}, \overline{\psi})$. More specifically, let $\mathcal{O}(C_i)(s_0) = \langle T_i, R_i\rangle$, where s_0 assigns to every channel-end the empty sequence (for notational convenience only, we assume that all channels are initially empty). Then either $\sigma{\downarrow_i} \in T_i$ or $(\sigma{\downarrow_i}, r_i) \in R_i$, for some set of input channels r_i on which the component C_i is blocked in the system state σ. By the validity of $I_i(z_i){:}\ C_i\{\psi_i\}$, we thus derive that either $\sigma{\downarrow_i}, \omega \models \psi_i$ or $\sigma{\downarrow_i}, \omega \models I_i(r_i)$. Moreover, since σ is a deadlock state, we have that $\sigma{\downarrow_i}, \omega \models I_i(r_i)$ implies that $\sigma, \omega \models \forall x \in r_i(buf(x) = \epsilon)$. Summarizing the above, and using Lemma 1, we conclude $\sigma, \omega \models \delta(\overline{I}, \overline{\psi})$. Similarly, it follows that every successfully terminating computation in π results in a final state σ such that $\sigma, \omega \models \bigwedge_i(\psi_i{\downarrow_i})$.

Theorem 1. *For every component based system $\pi = C_1 \parallel \cdots \parallel C_n$ we have that $\vdash \pi\{\psi\}$ implies $\models \pi\{\psi\}$.*

5.1 Completeness and Compositionality

The main rule of our proof system for deriving system-wide specifications allows compositional reasoning in terms of the interface specifications of the constituent components of a system. Therefore, completeness of the proof system semantically boils down to showing that the observable behavior of a system can be obtained in a compositional manner from the observable behavior of its components. Generally, although compositionality is a highly desirable property, it is not readily present in the formal models of component-based systems. In fact, our abstract semantics for components decouples the inherent ordering of the transmission and reception of values through different channels, and is not compositional in the general case. The following example illustrate this.

Consider the following three transition systems describing three different components (we omit all transitions derivable by the input reactiveness property).



It is not hard to see that all three components have the same observable behavior. However, consider a system consisting of one of these components together with the following one:

$$\cdot \xrightarrow{b?v} \cdot \xrightarrow{a!u} \cdot \xrightarrow{d?z} \cdot \xrightarrow{c!w} \cdot$$

If the system includes the component in the middle then it may deadlock, whereas deadlock is not possible if it includes the rightmost or leftmost component.

The above example shows two situations where compositionality breaks down, leading to violation of the completeness of our proof system. The crux of these counter-examples is that the environment is allowed to influence the nondeterministic behavior of a component. In order to prevent this, we need to identify the forms of external nondeterminism that must be forbidden, to obtain a compositional characterization of the observable behavior of a system in terms of the observable behavior of its components.

There are three reasons why a component may exhibit a nondeterministic behavior that can be resolved by the influence of the environment: (1) a nondeterministic choice involving input actions; (2) receiving a value from a channel-end shared with other components; and (3) sending a value to a channel-end shared with other components.

We rule out the first kind of external non-determinism by requiring a component to be *input confluent*. Formally, a component $C \widehat{=} \langle L, i, r, \longrightarrow \rangle$ is said to be input confluent if, for all $l \in L$,

1. if $l \xrightarrow{c?v} l_1$ and $l \xrightarrow{c'!v'} l_2$ then there exists $l' \in L$ such that $l_1 \xrightarrow{c'!v'} l'$ and $l_2 \xrightarrow{c?v} l'$;
2. if $l \xrightarrow{c?v} l_1$ and $l \xrightarrow{c?v} l_2$ then $l_1 = l_2$; and
3. if $l \xrightarrow{c?v} l_1$ and $l \xrightarrow{c'?v'} l_2$ with $c \neq c'$ then there exists $l' \in L$ such that $l_1 \xrightarrow{c'?v'} l'$ and $l_2 \xrightarrow{c?v} l'$.

In other words, a nondeterministic choice involving an input communication (on different channels) may delay that communication but cannot discharge it. Note that this is the case when different input actions are executed by parallel processes within a component. Returning to our counter-example, the left component violates the first condition and the middle component violates the second one.

To avoid the interference caused by the sharing of channel-ends among several components, we restrict to channels that are uni-directional and one-to-one. This means that every channel is an exclusive point-to-point communication medium between a single producer and a single consumer. The producer or the consumer of a channel must then lose its exclusive ownership of its end of a channel when it writes the name of that channel-end to a channel. Subsequently, a component may dynamically regain the exclusive ownership of a specific end of a channel, by reading its identity as a value from another channel. This way, the components in a system can dynamically reconfigure their channel connections.

A formal characterization of uni-directional and one-to-one channels is expressed in the two conditions below. We require that every component in a system is input confluent and that every computation $i \xrightarrow{a_1} l_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} l_n$ in every component $C \widehat{=} \langle L, i, r, \longrightarrow \rangle$ satisfies the following two conditions:

1. if $a_n = c?v$ then either $c \in r$ and $a_k \neq \overline{e}!c$, for all $1 \leq k < n$ and $e \in \mathit{Chan}$; or there exists $1 \leq i < j$ such that $a_i = d?c$, $d \neq c$, and $a_k \neq \overline{e}!c$ for all $j \leq k < n$ and $e \in \mathit{Chan}$.
2. if $a_n = \overline{c}!v$ then either $\overline{c} \in r$ and $a_k \neq \overline{e}!c$, for all $1 \leq k < n$ and $e \in \mathit{Chan}$; or there exists $1 \leq i < j$ such that $a_i = d?\overline{c}$ and $a_k \neq \overline{e}!c$, for all $j \leq k < n$ and $e \in \mathit{Chan}$.

Thus, a component can communicate via a channel-end only if (1) it has once been connected to the channel-end (either because the channel-end is included in the set of the initial connections of the component, or because the component has received the identity of the channel-end through a read action on another channel); and (2) the component has not subsequently relinquished its ownership of the channel-end by writing the identity of the channel-end to a channel.

We now show that the observable behavior of a system consisting of input confluent components (with exclusively point-to-point channels) can be described as a composition of the observable behavior of its components. Let

$\pi = C_1 \parallel \cdots \parallel C_n$ be a component-based system with $\mathcal{O}(C_k)(s_k) = \langle T_k, R_k \rangle$, $k \in \{0, \ldots, n\}$, as the semantics of its components. We define the set $\bigsqcup_k T_k$ of system states σ such that for every k there exists a component state $s'_k \in T_k$ with $\sigma(c) \downarrow_k = s'_k(c)$ and $\sigma(\overline{c}) \downarrow_k = s'_k(\overline{c})$, for every channel c. Recall that $\downarrow_k$ denotes the projection operation that, for a given sequence w of indexed values, yields the sequence of values indexed by k.

Similarly, we define the set $\bigsqcup_i \langle T_i, R_i \rangle$ of system states σ such that there is at least one k for which there exists a ready-set $(s'_k, r) \in R_k$ such that $\sigma(c) = \epsilon$ for every $c \in r$ and $\sigma(c)\downarrow_k = s'_k(c)$, and for every k for which this does not hold, there exists a component state $s'_k \in T_k$ with $\sigma(c)\downarrow_k = s'_k(c)$.

The following theorem states that for a system $\pi = C_1 \parallel \cdots \parallel C_n$ composed of input-confluent components connected only by point-to-point, uni-directional channels, we can describe the semantics of π, $\mathcal{O}(\pi)$, as a composition of the semantics of its components, $\mathcal{O}(C_k)$, $k = 1, \ldots, n$.

Theorem 2. *Let $\pi = C_1 \parallel \cdots \parallel C_n$ be as described above. Let σ be a system state and $s_k = \sigma \downarrow_k$, $k = 1, \ldots, n$. Given $\mathcal{O}(C_k)(s_k) = \langle T_k, R_k \rangle$, for $k \in \{1, \ldots, n\}$, we have*

$$\mathcal{O}(\pi)(\sigma) = \begin{cases} \bigsqcup_i T_i & \text{if } \bigsqcup_i \langle T_i, R_i \rangle = \emptyset \\ \delta & \text{otherwise.} \end{cases}$$

Assuming that we can express in the assertion language the observable behavior $\mathcal{O}(C)$ of a component C, we derive as a consequence of the above compositionality theorem the following (relative) completeness theorem.

Theorem 3. *For every component-based system $\pi = C_1 \parallel \cdots \parallel C_n$ where the behavior of every component $C_k \widehat{=} \langle L_k, i_k, r_k, \longrightarrow_k \rangle$ is input confluent, and components are connected only by point-to-point, uni-directional channels, we have that $\models \pi\{\psi\}$ if and only if $\vdash \pi\{\psi\}$.*

6 Conclusion and Future Work

The work reported in this paper is a further development of [2] and [7]. In [2] a language for dynamic networks of components is introduced, and in [7] a compositional semantics for its asynchronous subset is given. In this paper we abstract from the syntactical representation of a component and present a sound and complete description of a system in terms of the interfaces of its components.

For simplicity, in this paper we restricted the class of component-based systems by disallowing dynamic creation of components and channels. Our semantic framework, however, can easily be extended to relax these restrictions, as shown in [7]. Currently, we are investigating other forms of communication among components in systems that retain a compositional semantics with respect to our notion of observables.

We also intend to extend our proposed assertional language with features we borrow from temporal logic, in order to reason about the reactive behavior of a component.

Acknowledgements We like to thank the Amsterdam Coordination Group, especially Jaco de Bakker, Falk Bartels and Jerry den Hartog for discussions and suggestions about the contents of this paper. Thank also to Erika A'braham-Mumm for her helpful comments.

References

1. G. Agha, I. Mason, S. Smith, and C. Talcott. A foundation for actor computation *Journal of Functional Programming*, 1(1):1-69, 1993.
2. F. Arbab, M.M. Bonsangue, and F.S. de Boer. A coordination language for mobile components. In *Proc. of SAC 2000*, pages 166–173, ACM press, 2000.
3. F. Arbab, I. Herman, and P. Spilling. An overview of Manifold and its implementation. *Concurrency: Practice and Experience*, 5(1):23–70, 1993.
4. K. Bergner, A. Rausch, M. Sihling, A. Vilbig An integrated view on componentware: concepts, description techniques, and development process. In R. Lee, editor, *Proc. of IASTED Conference on Software Engineering*, pages 77–82, ACTA Press, 1998.
5. K. Bergner, A. Rausch, M. Sihling, A. Vilbig, and M. Broy. A formal model for componentware. In M. Sitaraman and G. Leavens, editors, *Foundation of Component-Based Systems*, Cambridge University Press, 2000.
6. F.S. de Boer. Reasoning about asynchronous communication in dynamically evolving object structures. In *Theoretical Computer Science*, 2000.
7. F.S. de Boer and M.M. Bonsangue. A compositional model for confluent dynamical data-flow networks. In B. Rovan ed., *Proc. 25th MFCS*, LNCS, 2000.
8. M.M. Bonsangue, F. Arbab, J.W. de Bakker, J.J.M.M. Rutten, A. Scutellá, and G. Zavattaro. A transition system semantics for the control-driven coordination language MANIFOLD. *Theoretical Computer Science*, 240(1), July 2000.
9. M. Broy. Equations for describing dynamic nets of communicating systems. In *Proc. 5th COMPASS workshop*, volume 906 of LNCS, pages 170–187, 1995.
10. L. Cardelli and A.D. Gordon. Mobile ambients. In *Proc. of Foundation of Software Science and Computational Structures*, volume 1378 of LNCS, pages 140-155, 1998.
11. R. Grosu and K. Stølen. A model for mobile point-to-point data-flow networks without channel sharing. In *Proc. AMAST'96*, LNCS, 1996.
12. JavaSoft. The JavaBeans component architecture, 1999. Available on line at the URL: `http://java.sun.com/beans`.
13. He Jifeng, M.B. Josephs, and C.A.R. Hoare. A theory of synchrony and asynchrony. In *Proc. of IFIP Working Conference on Programming Concepts and Methods*, pages 459-478, 1990.
14. Microsoft Corporation. ActiveX Controls, 1999 Available on line at the URL: `http://www.microsoft.com/com/tech/activex.asp`.
15. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts I and II. *Information and Computation* 100:1, 1992, pp. 1–77.
16. Object Management Group. CORBA 2.1 specifications, 1997. Available on line at the URL:`http://www.omg.org`.
17. E.-R. Olderog and C.A.R. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica* 23:9–66, 1986.
18. M. Shaw, R. De Line, D. Klein, T. Ross, D. Young and G. Zelesnik. Abstraction for software architectures and tools to support them. *IEEE Transactions on Software Engineering* 21(4):356–372, 1995.