University of Amsterdam
Programming Research Group

An Interface Group for Process Components

J.A. Bergstra
C.A. Middelburg

Report PRG0707 October 2007

J.A. Bergstra

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ Amsterdam
The Netherlands

tel. +31 20 525.7591
e-mail: janb@science.uva.nl

C.A. Middelburg

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ Amsterdam
The Netherlands

e-mail: kmiddelb@science.uva.nl

Programming Research Group Electronic Report Series

An Interface Group for Process Components*

J.A. Bergstra[1,2] and C.A. Middelburg[1]

[1] Programming Research Group, University of Amsterdam,
P.O. Box 41882, 1009 DB Amsterdam, the Netherlands
[2] Department of Philosophy, Utrecht University,
P.O. Box 80126, 3508 TC Utrecht, the Netherlands
J.A.Bergstra@uva.nl,C.A.Middelburg@uva.nl

Abstract. We take a process component as a pair of an interface and a behaviour. We study the composition of interacting process components in the setting of process algebra. We formalize the interfaces of interacting process components by means of an interface group. An interesting feature of the interface group is that it allows for distinguishing between expectations and promises in interfaces of process components. This distinction comes into play in case components with both client and server behaviour are involved.

Keywords: interface group, process component, process algebra.

1998 ACM Computing Classification: D.2.1, D.2.2, D.2.4, F.1.2, F.3.1.

1 Introduction

Component interfaces are a practical tool for the development of all but the most elementary architectural designs. In [7], interface groups have been proposed as a means to formalize the interfaces of the components of analytic execution architectures. The interface groups introduced in [7] concern component behaviours of two special kinds, called threads and services. The restriction to threads and services and the dichotomy between them are convenient in the case of analytic execution architectures, but inconvenient in the case of many other architectures. In this paper, we introduce an interface group which concerns behaviours of just one kind, namely processes as considered in the process algebra known as ACP [4, 9]. This gives rise to a more general setting because all threads and services can be looked upon as such processes (see e.g. [6]).

An interface group is a commutative group intended for describing and analysing interfaces. The interface group introduced in this paper concerns interfaces of process components that request other components to carry out methods and grant requests of other components to carry out methods. The interfaces in question represent the abilities to grant requests that are expected from other

* This research was partly carried out in the framework of the Jacquard-project Symbiosis, which is funded by the Netherlands Organisation for Scientific Research (NWO).

components and the abilities to make requests that are promised to other components. The ability to make a certain request and the ability to grant that request are considered to cancel out in interfaces. This allows among other things for establishing whether a system composed of a collection of process components is a closed system. Interfaces as modelled by the interface group introduced in this paper have less structure than the signatures used as interfaces in module algebra [3]. However, module algebra does not allow for distinguishing between expectations and promises in interfaces of components. In point of fact, it has a bias towards composing components whose interfaces concern promises only.

We also present a theory about process components of which the interface group introduced forms part. Like any notion of component, the notion of process component underlying this theory combines interface with content: a process component is considered a pair of an interface and a behaviour. Processes as considered in ACP are taken as the behaviours of process components. Therefore, the theory concerned is a development on top of ACP. However, additional assumptions are made about the actions of which the processes are made up. Three kinds of actions are distinguished: the requests referred to above, the grants referred to above, and the acts of carrying out a method which result from making a request and granting that request at the same time. The use of the presented theory about process components is illustrated by means of an example. A model of the theory is constructed, using a notion of bisimilarity for process components.

In the presented theory about process components, processes reside at places, called loci, and requests and grants are addressed to the processes residing at a certain locus. If the processes that are taken as the behaviours of process components are looked at in isolation, it may be convenient to abstract from the loci at which they reside. This abstraction gives rise to another kind of processes. We treat this kind of processes, referred to as localized processes, as well.

The structure of this paper is as follows. First, we review ACP (Section 2), review guarded recursion in the setting of ACP (Section 3), and present the actions that make up the processes being considered in later sections (Section 4). Next, we introduce a theory about integers (Section 5) and a theory about interfaces (Section 6). Then, we extend ACP, using the theories just introduced, to a theory about process components (Section 7). Following this, we give an example of the use of the presented theory about process components (Section 8). After that, we introduce a notion of bisimilarity for process components (Section 9) and construct a model of the presented theory about process components using this notion of bisimilarity (Section 10). Thereupon, we extend the theory about process components developed so far with localized processes (Section 11). Finally, we make some concluding remarks (Section 12).

2 Algebra of Communicating Processes

In this section, we shortly review ACP (Algebra of Communicating Processes), the algebraic theory about processes that was first presented in [4]. For a com-

prehensive overview of ACP, the reader is referred to [9]. Although ACP is one-sorted, we make this sort explicit. The reason for this is that we will extend ACP to a theory with four sorts in Section 7.

In ACP, it is assumed that a fixed but arbitrary finite set of *actions* $\mathcal{A}$, with $\delta \notin \mathcal{A}$, has been given. We write $\mathcal{A}_\delta$ for $\mathcal{A} \cup \{\delta\}$. It is further assumed that a fixed but arbitrary commutative and associative *communication* function $| : \mathcal{A}_\delta \times \mathcal{A}_\delta \to \mathcal{A}_\delta$, with $\delta \mid a = \delta$ for all $a \in \mathcal{A}_\delta$, has been given. The function $|$ is regarded to give the result of synchronously performing any two actions for which this is possible, and to be δ otherwise.

ACP has one sort: the sort $\mathbf{P}$ of *processes*. To build terms of sort $\mathbf{P}$, ACP has the following constants and operators:

- the *deadlock* constant $\delta : \mathbf{P}$;
- for each $a \in \mathcal{A}$, the *action* constant $a : \mathbf{P}$;
- the binary *alternative composition* operator $+ : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;
- the binary *sequential composition* operator $\cdot : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;
- the binary *parallel composition* operator $\parallel : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;
- the binary *left merge* operator $\lfloor\!\lfloor : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;
- the binary *communication merge* operator $| : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;
- for each $H \subseteq \mathcal{A}$, the unary *encapsulation* operator $\partial_H : \mathbf{P} \to \mathbf{P}$.

Terms of sorts $\mathbf{P}$ are built as usual for a one-sorted signature (see e.g. [15, 14]) Throughout the paper, we assume that there are infinitely many variables of sort $\mathbf{P}$, including x, y, z, x', y' and z'.

We use infix notation for the binary operators. The following precedence conventions are used to reduce the need for parentheses. The operator $+$ binds weaker than all other binary operators to build terms of sort $\mathbf{P}$ and the operator $\cdot$ binds stronger than all other binary operators to build terms of sort $\mathbf{P}$.

Let P and Q be closed terms of sort $\mathbf{P}$, $a \in \mathcal{A}$, and $H \subseteq \mathcal{A}$. Intuitively, the constants and operators to build terms of sort $\mathbf{P}$ can be explained as follows:

- δ can neither perform an action nor terminate successfully;
- a first performs action a and then terminates successfully;
- $P + Q$ behaves either as P or as Q, but not both;
- $P \cdot Q$ first behaves as P, but when P terminates successfully it continues by behaving as Q;
- $P \parallel Q$ behaves as the process that proceeds with P and Q in parallel;
- $P \lfloor\!\lfloor Q$ behaves the same as $P \parallel Q$, except that it starts with performing an action of P;
- $P \mid Q$ behaves the same as $P \parallel Q$, except that it starts with performing an action of P and an action of Q synchronously;
- $\partial_H(P)$ behaves the same as P, except that actions from H are blocked.

We write $\sum_{i \in \mathcal{I}} P_i$, where $\mathcal{I} = \{i_1, \ldots, i_n\}$ and $P_{i_1}, \ldots, P_{i_n}$ are terms of sort $\mathbf{P}$, for $P_{i_1} + \ldots + P_{i_n}$. The convention is that $\sum_{i \in \mathcal{I}} P_i$ stands for δ if $\mathcal{I} = \emptyset$.

The axioms of ACP are the axioms given in Table 1. CM2–CM3, CM5–CM7, C1–C3 and D1–D4 are actually axiom schemas in which a, b and c stand for

Table 1. Axioms of ACP

$x + y = y + x$		A1	$x \parallel y = x \between y + y \between x + x \mid y$	CM1
$(x + y) + z = x + (y + z)$		A2	$a \between x = a \cdot x$	CM2
$x + x = x$		A3	$a \cdot x \between y = a \cdot (x \parallel y)$	CM3
$(x + y) \cdot z = x \cdot z + y \cdot z$		A4	$(x + y) \between z = x \between z + y \between z$	CM4
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$		A5	$a \cdot x \mid b = (a \mid b) \cdot x$	CM5
$x + \delta = x$		A6	$a \mid b \cdot x = (a \mid b) \cdot x$	CM6
$\delta \cdot x = \delta$		A7	$a \cdot x \mid b \cdot y = (a \mid b) \cdot (x \parallel y)$	CM7
			$(x + y) \mid z = x \mid z + y \mid z$	CM8
			$x \mid (y + z) = x \mid y + x \mid z$	CM9
$\partial_H(a) = a$	if $a \notin H$	D1		
$\partial_H(a) = \delta$	if $a \in H$	D2	$a \mid b = b \mid a$	C1
$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$		D3	$(a \mid b) \mid c = a \mid (b \mid c)$	C2
$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$		D4	$\delta \mid a = \delta$	C3

arbitrary constants of sort **P** (keep in mind that also the deadlock constant belongs to the constants of sort **P**) and H stands for an arbitrary subset of $\mathcal{A}$.

For the main models of ACP, the reader is referred to [2].

3 Guarded Recursion

In this section, we shortly review guarded recursion in the setting of ACP.

Not all processes in a model of ACP have to be the interpretation of some closed term of sort **P**. Those processes may be definable over ACP.

A process in some model of ACP is *definable* over ACP if there exists a guarded recursive specification over ACP of which that process is the unique solution.

A *recursive specification* over ACP is a set of recursion equations $\{X = t_X \mid X \in V\}$ where V is a set of variables of sort **P** and each t_X is a term of sort **P** from the language of ACP that only contains variables from V. Let E be a recursive specification over ACP. Then we write $\mathrm{V}(E)$ for the set of all variables that occur on the left-hand side of an equation in E. A *solution* of a recursive specification E is a set of processes (in some model of ACP) $\{p_X \mid X \in \mathrm{V}(E)\}$ such that the equations of E hold if, for all $X \in \mathrm{V}(E)$, X stands for p_X.

Let t be a term of sort **P** from the language of ACP containing a variable X. Then an occurrence of X in t is *guarded* if t has a subterm of the form $a \cdot t'$ where $a \in \mathcal{A}$ and t' is a term containing this occurrence of X. Let E be a recursive specification over ACP. Then E is a *guarded recursive specification* if, in each equation $X = t_X \in E$, all occurrences of variables in t_X are guarded or t_X can be rewritten to such a term using the axioms of ACP in either direction and/or the equations in E except the equation $X = t_X$ from left to right. We

Table 2. Axioms for recursion

$\langle X\|E\rangle = \langle t_X\|E\rangle$	if $X = t_X \in E$	RDP
$E \Rightarrow X = \langle X\|E\rangle$	if $X \in \mathrm{V}(E)$	RSP

are only interested in models of ACP in which guarded recursive specifications have unique solutions.

For each guarded recursive specification E and each variable $X \in \mathrm{V}(E)$, we introduce a constant of sort $\mathbf{P}$ standing for the unique solution of E for X. This constant is denoted by $\langle X|E\rangle$.

The additional axioms for recursion are given in Table 2. In this table, we write $\langle t_X|E\rangle$ for t_X with, for all $Y \in \mathrm{V}(E)$, all occurrences of Y in t_X replaced by $\langle Y|E\rangle$. Both RDP and RSP are axiom schemas. Side conditions are added to restrict the variables, terms and guarded recursive specifications for which X, t_X and E stand. RDP and RSP were first formulated in [5].

We write ACP+REC for ACP extended with the constants standing for the unique solutions of guarded recursive specifications and the axioms RDP and RSP.

4 ACP for Cooperating Components

In this paper, we consider process components that cooperate by making and granting requests to carry out methods. The processes that are taken as the behaviours of these components are not made up of arbitrary actions. In this section, we introduce the instance of ACP that is restricted to the intended actions. This instance is called $\mathrm{ACP}_{\mathrm{CC}}$ (ACP for Cooperating Components).

Three kinds of actions are distinguished in $\mathrm{ACP}_{\mathrm{CC}}$: active actions, passive actions and neutral actions. The active actions may be viewed as requests to carry out some method and the passive actions may be viewed as grants of requests to carry out some method. Making a request to carry out some method and granting that request at the same time results in carrying out the method concerned. The initiative in carrying out the method is considered to be taken by the process making the request. This explains why the request is called an active action and its grant is called a passive action. The neutral actions may be viewed as the results of making a request to carry out some method and granting that request at the same time. A process that can perform active actions only behaves as a client and a process that can perform passive actions only behaves as a server.

In $\mathrm{ACP}_{\mathrm{CC}}$, it is assumed that a fixed but arbitrary finite set $\mathcal{L}$ of *loci* and a fixed but arbitrary finite set $\mathcal{M}$ of *methods* have been given. A locus is a place at which processes reside. Intuitively, a process resides at a locus if it is capable of performing actions in that locus. The same process may reside at different loci at once. Moreover, different processes may reside at the same locus at once. Therefore, we do not necessarily refer to a unique process if we refer to a process residing at a given locus.

In $\mathrm{ACP}_{\mathrm{CC}}$, the set of actions $\mathcal{A}$ consists of:

- for each $f, g \in \mathcal{L}$ and $m \in \mathcal{M}$, the *active action* $f.m@g$;
- for each $f, g \in \mathcal{L}$ and $m \in \mathcal{M}$, the *passive action* $\sim f.m@g$;
- for each $f, g \in \mathcal{L}$ and $m \in \mathcal{M}$, the *neutral action* $f.m\underline{@}g$.

Intuitively, these actions can be explained as follows:

- $f.m@g$ is the action by which a process residing at locus g requests a process residing at locus f to carry out method m;
- $\sim g.m@f$ is the action by which a process residing at locus f grants a request of a process residing at locus g to carry out method m;
- $f.m\underline{@}g$ is the result of performing $f.m@g$ and $\sim g.m@f$ at the same time.

In $\mathrm{ACP}_{\mathrm{CC}}$, the communication function $| : \mathcal{A}_\delta \times \mathcal{A}_\delta \to \mathcal{A}_\delta$ is such that for all $f, g \in \mathcal{L}$ and $m \in \mathcal{M}$:

- $f.m@g \mid \sim g.m@f = f.m\underline{@}g$;
- $f.m@g \mid a = \delta$ for all $a \in \mathcal{A} \setminus \{\sim g.m@f\}$;
- $a \mid \sim g.m@f = \delta$ for all $a \in \mathcal{A} \setminus \{f.m@g\}$;
- $f.m\underline{@}g \mid a = \delta$ for all $a \in \mathcal{A}$.

The receive actions and send actions commonly used for handshaking communication of data, see e.g. [2], can be viewed as requests to carry out some communication method and grants of such requests, respectively. However, the current set-up requires that it is made explicit what are the loci at which the sender and receiver involved reside.

5 Integers

In this section, we present an algebraic theory about integers which will be used in later sections. The presented theory is called INT.

INT has one sort: the sort $\mathbf{Z}$ of *integers*. To build terms of sort $\mathbf{Z}$, INT has the following constants and operators:

- the constant $0 : \mathbf{Z}$;
- the constant $1 : \mathbf{Z}$;
- the binary *addition* operator $+ : \mathbf{Z} \times \mathbf{Z} \to \mathbf{Z}$;
- the unary *additive inverse* operator $- : \mathbf{Z} \to \mathbf{Z}$;
- the unary *signum* operator $\mathsf{sg} : \mathbf{Z} \to \mathbf{Z}$.

Terms of sort $\mathbf{Z}$ are built as usual for a one-sorted signature. Throughout the paper, we assume that there are infinitely many variables of sort $\mathbf{Z}$, including k, l and n.

As usual, we use infix notation for the binary operator $+$ and prefix notation for the unary operator $-$. The following additional precedence convention is used to reduce the need for parentheses. The operator $+$ binds weaker than the operator $-$.

Table 3. Axioms of INT

$0 + k = k$	INT1
$-k + k = 0$	INT2
$(k + l) + n = k + (l + n)$	INT3
$k + l = l + k$	INT4
$\mathsf{sg}(0) = 0$	SG1
$\mathsf{sg}(1) = 1$	SG2
$\mathsf{sg}(-1) = -1$	SG3
$\mathsf{sg}(k + \mathsf{sg}(k)) = \mathsf{sg}(k)$	SG4

The constants and operators of INT are adopted from integer arithmetic and need no further explanation. The operator sg is useful where a distinction between positive integers, negative integers and zero must be made.

The axioms of INT are the axioms given in Table 3. Axioms INT1–INT4 are the axioms of a commutative group. Axioms SG1–SG4 are the defining axioms of sg.

The initial model of INT is considered the standard model of INT.

6 Interface Group for Cooperating Components

In this section, we present an algebraic theory about interfaces. The presented theory is called called $\mathrm{IFG}_{\mathrm{CC}}$. In Section 7, we will consider process components which are taken as pairs of an interface and a process that is made up of active action, passive actions, and neutral actions. Interfaces are built from two kinds of interface elements.

The set of *interface elements* consists of:

- for each $f, g \in \mathcal{L}$ and $m \in \mathcal{M}$, the *active interface element* $f.m@g$;
- for each $f, g \in \mathcal{L}$ and $m \in \mathcal{M}$, the *passive interface element* $\sim f.m@g$.

We write $\mathcal{IFE}$ for the set of all interface elements.

Obviously, $\mathcal{IFE}$ is a proper subset of $\mathcal{A}$. The interface elements are those actions that are allowed to occur in interfaces. The interface part of a process component consist of the active and passive actions that the process part of that process component may be capable of performing. The interface elements $f.m@g$ and $\sim g.m@f$ are considered each other inverses. That is, if both occur in an interface, they cancel out.

Active interface elements are usually included in the interface of a process component to couch that it expects from the environment in which it is put the ability to grant certain requests. Passive interface elements are usually included in the interface of a process component to couch that it promises the environment in which it is put the ability to make certain requests. The environment in which the process component is put may be composed of different process components.

To couch that it expects from a number of process components an ability or it promises a number of process components an ability, the relevant interface element is included the number of times concerned in the interface of the process component.

The distinction between active interface elements and passive interface elements made here is a case of distinction between expectations and promises because interface elements are actions that process components may be capable of performing. If the interface elements would be actions that process components must be capable of performing, it would be a case of distinction between requirements and provisions.

Interfaces can be considered multisets over the set of all active interface elements in which multiplicities of elements may be negative too, since occurrences of passive interface elements in an interface can be counted as negative occurrences of their inverses.

$\mathrm{IFG}_{\mathrm{CC}}$ has the sort $\mathbf{Z}$ from INT and in addition the sort $\mathbf{I}$ of *interfaces*. To build terms of sort $\mathbf{I}$, $\mathrm{IFG}_{\mathrm{CC}}$ has the following constants and operators:

- the *empty interface* constant $0 : \mathbf{I}$;
- for each $e \in \mathcal{IFE}$, the *interface element* constant $e : \mathbf{I}$;
- the binary *interface combination* operator $+ : \mathbf{I} \times \mathbf{I} \to \mathbf{I}$;
- the unary *interface inversion* operator $- : \mathbf{I} \to \mathbf{I}$.

To build terms of sort $\mathbf{Z}$, $\mathrm{IFG}_{\mathrm{CC}}$ has the constants and operators of INT and in addition the following operator:

- for each $f, g \in \mathcal{L}$ and $m \in \mathcal{M}$, the unary *multiplicity* operator $\#_{f.m@g} : \mathbf{I} \to \mathbf{Z}$.

Terms of the sorts $\mathbf{I}$ and $\mathbf{Z}$ are built as usual for a many-sorted signature (see e.g. [15, 14]). Throughout the paper, we assume that there are infinitely many variables of sort $\mathbf{I}$, including i, j and h.

We use infix notation for the binary operator $+$ and prefix notation for the unary operator $-$. The following precedence convention is used to reduce the need for parentheses. The operator $+$ binds weaker than the operator $-$.

Let I and J be closed terms of sort $\mathbf{I}$, $f, g \in \mathcal{L}$, and $m \in \mathcal{M}$. Viewing interfaces as multisets with multiplicities from $\mathbf{Z}$, the constants and operators of $\mathrm{IFG}_{\mathrm{CC}}$ to build terms of sort $\mathbf{I}$ can be explained as follows:

- 0 is the interface in which the multiplicity of each active interface element is 0;
- $f.m@g$ is the interface in which the multiplicity of $f.m@g$ is 1 and the multiplicity of each other active interface element is 0;
- $\sim f.m@g$ is the interface in which the multiplicity of $g.m@f$ is -1 and the multiplicity of each other active interface element is 0;
- $I + J$ is the interface in which the multiplicity of each active interface element is the addition of its multiplicities in I and J;
- $-I$ is the interface in which the multiplicity of each active interface element is the additive inverse of its multiplicity in I.

Table 4. Axioms of $\mathrm{IFG}_{\mathrm{CC}}$

$0 + i = i$	IFG1
$-i + i = 0$	IFG2
$(i + j) + h = i + (j + h)$	IFG3
$i + j = j + i$	IFG4
$f.m@g + {\sim}g.m@f = 0$	IFG5
$\#_{f.m@g}(0) = 0$	M1
$\#_{f.m@g}(f'.m'@g') = 0 \quad \mathsf{if}\ f \neq f' \vee m \neq m' \vee g \neq g'$	M2
$\#_{f.m@g}(f.m@g) = 1$	M3
$\#_{f.m@g}(-i) = -\#_{f.m@g}(i)$	M4
$\#_{f.m@g}(i + j) = \#_{f.m@g}(i) + \#_{f.m@g}(j)$	M5

The operators $\#_{f.m@g}$, one for each $f, g \in \mathcal{L}$ and $m \in \mathcal{M}$, can simply be explained as follows:

- $\#_{f.m@g}(I)$ is the multiplicity of $f.m@g$ in I.

We write $\sum_{i \in \mathcal{I}} I_i$, where $\mathcal{I} = \{i_1, \ldots, i_n\}$ and $I_{i_1}, \ldots, I_{i_n}$ are terms of sort $\mathbf{I}$, for $I_{i_1} + \ldots + I_{i_n}$. The convention is that $\sum_{i \in \mathcal{I}} I_i$ stands for 0 if $\mathcal{I} = \emptyset$.

The axioms of $\mathrm{IFG}_{\mathrm{CC}}$ are the axioms of INT and the axioms given in Table 4. IFG5 and M1–M5 are actually axiom schemas in which f and g stand for arbitrary members of $\mathcal{L}$ and m stands for an arbitrary member of $\mathcal{M}$. Axioms IFG1–IFG4 are the axioms of a commutative group and axiom IFG5, called the *reflection law*, states that ${\sim}g.m@f$ is taken as the inverse of $f.m@g$. Axioms M1–M5 are the defining axioms of $\#_{f.m@g}$.

The initial model of $\mathrm{IFG}_{\mathrm{CC}}$ is considered the standard model of $\mathrm{IFG}_{\mathrm{CC}}$.

7 Algebra of Cooperating Components

In this section, we take up the extension of $\mathrm{ACP}_{\mathrm{CC}}$ to a theory about process components. The result is called ACC (Algebra of Cooperating Components).

In the preceding sections, we have already been gone into some of the general ideas that underlie the design of this extension. Those ideas, which concern the interfaces and behaviours of process components, can be summarized as follows:

- behaviours of process components are processes made up of three kinds of actions: active actions, passive actions and neutral actions;
- for each active action, there is a unique passive action with which it can be performed synchronously, and vice versa;
- interfaces of process components consist of active and passive actions that the process components may be capable of performing;
- looked upon as an interface element, each active action has the passive action with which it can be performed synchronously as its inverse, and vice versa;

– in interfaces of process components, there may be elements with multiple occurrences.

The remaining general ideas concern the process components by themselves:

– if a process is turned into a process component by adding an interface to it, the process is restricted by the interface with respect to the active and passive actions that it can perform to force that the behaviour of the process component complies with its interface;
– if two process components are composed, the interface of the composed process component is the combination of the interfaces of the two process components and the behaviour of the composed process component is the parallel composition of the behaviours of the two process components restricted by the combination of the interfaces of the two process components.

The point of view on the composition of process components implies that, if all occurrences of an (active or passive) action in the interface of a process component are cancelled out by composition with another process component, this action is blocked in the behaviour of the composition of these process components. The blocking of the action takes place even if its inverse is not included in the actions that make up the behaviour of the other process component. It is possible that the inverse is not included because the interfaces concern expectations and promises instead of requirements and provisions (see also Section 6). The way in which is dealt with this possibility can be explained as follows: (i) if a promised ability to make a request is not provided, making the request is blocked and (ii) if an expected ability to grant a request is not required, granting the request is blocked.

ACC has the sort $\mathbf{P}$ from $\mathrm{ACP}_{\mathrm{CC}}$, the sorts $\mathbf{I}$ and $\mathbf{Z}$ from $\mathrm{IFG}_{\mathrm{CC}}$, and in addition the sort $\mathbf{C}$ of *components*. To build terms of sort $\mathbf{C}$, ACC has the following operators:

– the binary *basic component* operator $\mathsf{c} : \mathbf{I} \times \mathbf{P} \to \mathbf{C}$;
– the binary *component composition* operator $\| : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$.

To build terms of sort $\mathbf{P}$, ACC has the constants and operators of $\mathrm{ACP}_{\mathrm{CC}}$ and in addition the following operator:

– the binary *interface compliant encapsulation* operator $\overline{\partial} : \mathbf{I} \times \mathbf{P} \to \mathbf{P}$.

To build terms of sort $\mathbf{I}$, ACC has the constants and operators of $\mathrm{IFG}_{\mathrm{CC}}$ to build terms of sort $\mathbf{I}$. To build terms of sort $\mathbf{Z}$, ACC has the constants and operators of $\mathrm{IFG}_{\mathrm{CC}}$ to build terms of sort $\mathbf{Z}$.

Terms of the different sorts are built as usual for a many-sorted signature. Throughout the paper, we assume that there are infinitely many variables of sort $\mathbf{C}$, including u, v, u' and v'.

We use infix notation for the binary operator $\|$. We write $\overline{\partial}_I(P)$, where I is a term of sort $\mathbf{I}$ and P is a term of sort $\mathbf{P}$, for $\overline{\partial}(I, P)$.

Let C and D be closed terms of sort $\mathbf{C}$, P be a closed term of sort $\mathbf{P}$, and I be a closed term of sort $\mathbf{I}$. Viewing interfaces as multisets with multiplicities from $\mathbf{Z}$, the operators of ACC to build terms of sort $\mathbf{C}$ can be explained as follows:

Table 5. Axioms of ACC

$\mathsf{c}(i,x) = \mathsf{c}(i,\overline{\partial}_i(x))$	CC1
$\mathsf{c}(i,x) \parallel \mathsf{c}(j,y) = \mathsf{c}(i+j,\overline{\partial}_i(x) \parallel \overline{\partial}_j(y))$	CC2
$\mathsf{sg}(\#_{f.m@g}(i)) = 1 \Rightarrow \overline{\partial}_i(f.m@g) = f.m@g$	E1
$\mathsf{sg}(\#_{f.m@g}(i)) = 0 \Rightarrow \overline{\partial}_i(f.m@g) = \delta$	E2
$\mathsf{sg}(\#_{f.m@g}(i)) = -1 \Rightarrow \overline{\partial}_i(f.m@g) = \delta$	E3
$\mathsf{sg}(\#_{g.m@f}(i)) = -1 \Rightarrow \overline{\partial}_i(\sim f.m@g) = \sim f.m@g$	E4
$\mathsf{sg}(\#_{g.m@f}(i)) = 0 \Rightarrow \overline{\partial}_i(\sim f.m@g) = \delta$	E5
$\mathsf{sg}(\#_{g.m@f}(i)) = 1 \Rightarrow \overline{\partial}_i(\sim f.m@g) = \delta$	E6
$\overline{\partial}_i(f.m\underline{@}g) = f.m\underline{@}g$	E7
$\overline{\partial}_i(\delta) = \delta$	E8
$\overline{\partial}_i(x+y) = \overline{\partial}_i(x) + \overline{\partial}_i(y)$	E9
$\overline{\partial}_i(x \cdot y) = \overline{\partial}_i(x) \cdot \overline{\partial}_i(y)$	E10

- $\mathsf{c}(I,P)$ is the process component of which the interface is I and the behaviour is P, except that active actions of which the multiplicity in I is not positive and passive actions with an inverse of which the multiplicity in I is not negative are blocked;
- $C \parallel D$, is the process component of which the interface is the combination of the interfaces of C and D and the behaviour is the parallel composition of the behaviours of C and D, except that active actions of which the multiplicity in the combination of the interfaces of C and D is not positive and passive actions with an inverse of which the multiplicity in the combination of the interfaces of C and D is not negative are blocked.

The operator $\overline{\partial}$ can be explained as follows:

- $\overline{\partial}_I(P)$ behaves the same as P, except that active actions of which the multiplicity in I is not positive and passive actions with an inverse of which the multiplicity in I is not negative are blocked.

The operator $\overline{\partial}$ is an auxiliary operator used in the axioms concerning process components.

The axioms of ACC are the axioms of ACP, the axioms of $\mathrm{IFG}_{\mathrm{CC}}$, and the axioms given in Table 5. E1–E7 are actually axiom schemas in which f and g stand for arbitrary members of $\mathcal{L}$ and m stands for an arbitrary member of $\mathcal{M}$. Axioms CC1 and CC2 are axioms concerning process components and axioms E1–E10 are the defining axioms of the auxiliary operator $\overline{\partial}$. Together they formalize the intuition about process components given above in a direct way. It is only because they are used in axioms E1–E6 that the multiplicity operators $\#_{f.m@g}$ are included in the theory $\mathrm{IFG}_{\mathrm{CC}}$ and the signum operator sg is included in the theory INT.

Using the axioms of ACC, it can be proved that the actions that make up the behaviour of a process component $\mathsf{c}(I,P)$ include neither active actions nor

passive actions if $I = 0$. This fact makes a justification for identifying closed systems with process components that have an empty interface.

Guarded recursion can be added to ACC as it is added to ACP in Section 3. We write ACC+REC for ACC extended with the constants standing for the unique solutions of guarded recursive specifications and the axioms RDP and RSP.

In Section 10, we will construct a model of ACC+REC using a notion of bisimilarity for process components.

8 An Example

In this section, we illustrate the use of ACC by means of an example concerning buffers with capacity one. We assume a finite set $\mathcal{D}$ of data with $\mathrm{e} \in \mathcal{D}$ and, for each $d \in \mathcal{D}$, a method c_d for communicating datum d. We take the element $\mathrm{e} \in \mathcal{D}$ for an improper datum.

We consider the three buffer processes P_f, P_g, and P_h that are defined by the guarded recursion equations

$$\begin{aligned}
X_f &= \sum_{d \in \mathcal{D} \setminus \{\mathrm{e}\}} {\sim}s.c_d@f \cdot (g.c_d@f + g.c_{\mathrm{e}}@f) \cdot X_f \;, \\
X_g &= \sum_{d \in \mathcal{D} \setminus \{\mathrm{e}\}} {\sim}f.c_d@g \cdot (h.c_d@g + h.c_{\mathrm{e}}@g) \cdot X_g \;, \\
X_h &= \sum_{d \in \mathcal{D} \setminus \{\mathrm{e}\}} {\sim}g.c_d@h \cdot (r.c_d@h + r.c_{\mathrm{e}}@h) \cdot X_h \;,
\end{aligned}$$

respectively. The processes P_f, P_g and P_h always reside at the loci f, g and h, respectively. P_f is able to pass data from processes residing at locus s to processes residing at locus g, P_g is able to pass data from processes residing at locus f to processes residing at locus h, and P_h is able to pass data from processes residing at locus g to processes residing at locus r. P_f, P_g and P_h are faulty in the sense that they may deliver an improper datum instead of the datum to be delivered.

We turn these three buffer processes into process components by adding interfaces to them. To be exact, we turn the processes P_f, P_g, and P_h into the process components $\mathsf{c}(I_f, P_f)$, $\mathsf{c}(I_g, P_g)$, and $\mathsf{c}(I_h, P_h)$, where

$$\begin{aligned}
I_f &= \sum_{d \in \mathcal{D} \setminus \{\mathrm{e}\}} {\sim}s.c_d@f + \sum_{d \in \mathcal{D}} g.c_d@f \;, \\
I_g &= \sum_{d \in \mathcal{D} \setminus \{\mathrm{e}\}} {\sim}f.c_d@g + \sum_{d \in \mathcal{D}} h.c_d@g \;, \\
I_h &= \sum_{d \in \mathcal{D} \setminus \{\mathrm{e}\}} {\sim}g.c_d@h + \sum_{d \in \mathcal{D}} r.c_d@h \;.
\end{aligned}$$

We have a look at the component composition $\mathsf{c}(I_f, P_f) \parallel \mathsf{c}(I_g, P_g) \parallel \mathsf{c}(I_h, P_h)$. It follows from axiom CC2 that

$$\begin{aligned}
&\mathsf{c}(I_f, P_f) \parallel \mathsf{c}(I_g, P_g) \parallel \mathsf{c}(I_h, P_h) \\
&\quad = \mathsf{c}\big(I_f + I_g + I_h, \overline{\partial}_{I_f}(P_f) \parallel \overline{\partial}_{I_g}(P_g) \parallel \overline{\partial}_{I_h}(P_h)\big) \;.
\end{aligned}$$

Moreover, it follows from axioms IFG1–IFG5 that

$$I_f + I_g + I_h = \sum_{d \in \mathcal{D} \setminus \{\mathrm{e}\}} {\sim}s.c_d@f + g.c_\mathrm{e}@f + h.c_\mathrm{e}@g + \sum_{d \in \mathcal{D}} r.c_d@h$$

and from axioms INT1–INT4, SG1–SG4, M1–M5, and E1–E10 that

$$\overline{\partial}_{I_f}(P_f) \parallel \overline{\partial}_{I_g}(P_g) \parallel \overline{\partial}_{I_h}(P_h) = P_f \parallel P_g \parallel P_h \ .$$

Hence, we have that

$$\begin{aligned}&\mathsf{c}(I_f, P_f) \parallel \mathsf{c}(I_g, P_g) \parallel \mathsf{c}(I_h, P_h)\\ &\quad = \mathsf{c}\left(\sum_{d \in \mathcal{D} \setminus \{\mathrm{e}\}} {\sim}s.c_d@f + g.c_\mathrm{e}@f + h.c_\mathrm{e}@g + \sum_{d \in \mathcal{D}} r.c_d@h, P_f \parallel P_g \parallel P_h\right) .\end{aligned}$$

It can further be shown by means of the axioms of ACP that $P_f \parallel P_g \parallel P_h$, i.e. the behaviour of $\mathsf{c}(I_f, P_f) \parallel \mathsf{c}(I_g, P_g) \parallel \mathsf{c}(I_h, P_h)$, is essentially a buffer with capacity three. This buffer process, which resides alternately at the loci f, g and h, is able to pass data from processes residing at locus s to processes residing at locus r. It is faulty in the sense that it may deliver an improper datum instead of the datum to be delivered. Moreover, the improper datum may be delivered at the locus g or the locus h instead of the locus r.

A closed system is a process component that has an empty interface. Clearly, $\mathsf{c}(I_f, P_f) \parallel \mathsf{c}(I_g, P_g) \parallel \mathsf{c}(I_h, P_h)$ is not a closed system. It follows from axioms IFG1–IFG5 that composing it with a process component whose interface is

$$\sum_{d \in \mathcal{D} \setminus \{\mathrm{e}\}} f.c_d@s + {\sim}f.c_\mathrm{e}@g + {\sim}g.c_\mathrm{e}@h + \sum_{d \in \mathcal{D}} {\sim}h.c_d@r$$

would result in a closed system. This shows that a closed system requires composition with a process component that promises to handle the delivery of an improper datum at the loci g, h and r.

9 Bisimilarity of Process Components

In this section, we give a structural operational semantics for ACC+REC and define a notion of bisimilarity based on it. This notion of bisimilarity will be used in Section 10 to construct a model of ACC+REC.

Henceforth, we will write $\mathcal{T}_S$, where $S \in \{\mathbf{P}, \mathbf{I}, \mathbf{C}\}$, for the set of all closed terms of sort S from the language of ACC+REC. Moreover, we will write $\mathcal{T}_{\mathbf{Z}}^{\mathrm{INT}}$ for the set of all closed terms of sort $\mathbf{Z}$ from the language of INT.

The following relations are the primary relations used in the structural operational semantics of ACC+REC:

- a unary relation $\xrightarrow{a}_{\mathsf{p}} \surd \subseteq \mathcal{T}_{\mathbf{P}}$, for each $a \in \mathcal{A}$;
- a binary relation $\xrightarrow{a}_{\mathsf{p}} \subseteq \mathcal{T}_{\mathbf{P}} \times \mathcal{T}_{\mathbf{P}}$, for each $a \in \mathcal{A}$;
- a unary relation $f.m@g{\sqsubseteq^N} \subseteq \mathcal{T}_{\mathbf{I}}$, for each $f, g \in \mathcal{L}$, $m \in \mathcal{M}$ and $N \in \mathcal{T}_{\mathbf{Z}}^{\mathrm{INT}}$;

– a binary relation $\mathsf{hasIF} \subseteq \mathcal{T}_{\mathbf{C}} \times \mathcal{T}_{\mathbf{I}}$;
– a unary relation $\xrightarrow{a}_{\mathsf{c}} \surd \subseteq \mathcal{T}_{\mathbf{C}}$, for each $a \in \mathcal{A}$;
– a binary relation $\xrightarrow{a}_{\mathsf{c}} \subseteq \mathcal{T}_{\mathbf{C}} \times \mathcal{T}_{\mathbf{C}}$, for each $a \in \mathcal{A}$.

We write $P \xrightarrow{a}_{\mathsf{p}} \surd$ instead of $P \in \xrightarrow{a}_{\mathsf{p}} \surd$, $P \xrightarrow{a}_{\mathsf{p}} P'$ instead of $(P, P') \in \xrightarrow{a}_{\mathsf{p}}$, $f.m@g \sqsubseteq^N I$ instead of $I \in f.m@g\sqsubseteq^N$, $C\ \mathsf{hasIF}\ I$ instead of $(C, I) \in \mathsf{hasIF}$, $C \xrightarrow{a}_{\mathsf{c}} \surd$ instead of $C \in \xrightarrow{a}_{\mathsf{c}} \surd$, and $C \xrightarrow{a}_{\mathsf{c}} C'$ instead of $(C, C') \in \xrightarrow{a}_{\mathsf{c}}$. The relations can be explained as follows:

– $P \xrightarrow{a}_{\mathsf{p}} \surd$: process P is capable of first performing a and then terminating successfully;
– $P \xrightarrow{a}_{\mathsf{p}} P'$: process P is capable of first performing a and then proceeding as process P';
– $f.m@g \sqsubseteq^N I$: $f.m@g$ occurs N times in interface I;
– $C\ \mathsf{hasIF}\ I$: the interface of component C is I;
– $C \xrightarrow{a}_{\mathsf{c}} \surd$: component C is capable of first performing a and then terminating successfully;
– $C \xrightarrow{a}_{\mathsf{c}} C'$: component C is capable of first performing a and then proceeding as component C'.

The following relations are auxiliary relations used in the structural operational semantics of ACC+REC:

– a unary relation $f.m@g\sqsubseteq^+ \subseteq \mathcal{T}_{\mathbf{I}}$, for each $f, g \in \mathcal{L}$ and $m \in \mathcal{M}$;
– a unary relation $f.m@g\sqsubseteq^- \subseteq \mathcal{T}_{\mathbf{I}}$, for each $f, g \in \mathcal{L}$ and $m \in \mathcal{M}$;
– a unary relation $f.m@g\sqsubseteq^+ \mathsf{IF} \subseteq \mathcal{T}_{\mathbf{C}}$, for each $f, g \in \mathcal{L}$ and $m \in \mathcal{M}$;
– a unary relation $f.m@g\sqsubseteq^- \mathsf{IF} \subseteq \mathcal{T}_{\mathbf{C}}$, for each $f, g \in \mathcal{L}$ and $m \in \mathcal{M}$.

We write $f.m@g \sqsubseteq^+ I$ and $f.m@g \sqsubseteq^- I$ instead of $I \in f.m@g\sqsubseteq^+$ and $I \in f.m@g\sqsubseteq^-$, respectively. We write $f.m@g \sqsubseteq^+ \mathsf{IF}(I)$ and $f.m@g \sqsubseteq^- \mathsf{IF}(I)$ instead of $I \in f.m@g \sqsubseteq^+ \mathsf{IF}$ and $I \in f.m@g \sqsubseteq^- \mathsf{IF}$, respectively. The relations can be explained as follows:

– $f.m@g \sqsubseteq^+ I$: $f.m@g$ occurs a positive number of times in interface I;
– $f.m@g \sqsubseteq^- I$: $f.m@g$ occurs a negative number of times in interface I;
– $f.m@g \sqsubseteq^+ \mathsf{IF}(C)$: $f.m@g$ occurs a positive number of times in the interface of component C;
– $f.m@g \sqsubseteq^- \mathsf{IF}(C)$: $f.m@g$ occurs a negative number of times in the interface of component C.

The auxiliary relations are for convenience only.

The structural operational semantics of ACC+REC is described by the rules given in Tables 6 and 7.

The following uniqueness property of the relations $f.m@g\sqsubseteq^N$ will be used in Section 10 to construct a model of ACC+REC.

Lemma 1. *Let $f, g \in \mathcal{L}$ and $m \in \mathcal{M}$. Then for all $I \in \mathcal{T}_{\mathbf{I}}$, there exists an $N \in \mathcal{T}_{\mathbf{Z}}^{\mathrm{INT}}$ such that for all $N' \in \mathcal{T}_{\mathbf{Z}}^{\mathrm{INT}}$ with $f.m@g \sqsubseteq^{N'} I$ we have that $N = N'$ holds in the initial model of* INT.

Table 6. Rules for operational semantics of ACP+REC

$$\frac{}{a \xrightarrow{a}_{\mathsf{p}} \surd}$$

$$\frac{x \xrightarrow{a}_{\mathsf{p}} \surd}{x + y \xrightarrow{a}_{\mathsf{p}} \surd} \quad \frac{y \xrightarrow{a}_{\mathsf{p}} \surd}{x + y \xrightarrow{a}_{\mathsf{p}} \surd} \quad \frac{x \xrightarrow{a}_{\mathsf{p}} x'}{x + y \xrightarrow{a}_{\mathsf{p}} x'} \quad \frac{y \xrightarrow{a}_{\mathsf{p}} y'}{x + y \xrightarrow{a}_{\mathsf{p}} y'}$$

$$\frac{x \xrightarrow{a}_{\mathsf{p}} \surd}{x \cdot y \xrightarrow{a}_{\mathsf{p}} y} \quad \frac{x \xrightarrow{a}_{\mathsf{p}} x'}{x \cdot y \xrightarrow{a}_{\mathsf{p}} x' \cdot y}$$

$$\frac{x \xrightarrow{a}_{\mathsf{p}} \surd}{x \parallel y \xrightarrow{a}_{\mathsf{p}} y} \quad \frac{y \xrightarrow{a}_{\mathsf{p}} \surd}{x \parallel y \xrightarrow{a}_{\mathsf{p}} x} \quad \frac{x \xrightarrow{a}_{\mathsf{p}} x'}{x \parallel y \xrightarrow{a}_{\mathsf{p}} x' \parallel y} \quad \frac{y \xrightarrow{a}_{\mathsf{p}} y'}{x \parallel y \xrightarrow{a}_{\mathsf{p}} x \parallel y'}$$

$$\frac{x \xrightarrow{a}_{\mathsf{p}} \surd,\ y \xrightarrow{b}_{\mathsf{p}} \surd}{x \parallel y \xrightarrow{c}_{\mathsf{p}} \surd}\ a \mid b = c \quad \frac{x \xrightarrow{a}_{\mathsf{p}} \surd,\ y \xrightarrow{b}_{\mathsf{p}} y'}{x \parallel y \xrightarrow{c}_{\mathsf{p}} y'}\ a \mid b = c$$

$$\frac{x \xrightarrow{a}_{\mathsf{p}} x',\ y \xrightarrow{b}_{\mathsf{p}} \surd}{x \parallel y \xrightarrow{c}_{\mathsf{p}} x'}\ a \mid b = c \quad \frac{x \xrightarrow{a}_{\mathsf{p}} x',\ y \xrightarrow{b}_{\mathsf{p}} y'}{x \parallel y \xrightarrow{c}_{\mathsf{p}} x' \parallel y'}\ a \mid b = c$$

$$\frac{x \xrightarrow{a}_{\mathsf{p}} \surd}{x \between y \xrightarrow{a}_{\mathsf{p}} y} \quad \frac{x \xrightarrow{a}_{\mathsf{p}} x'}{x \between y \xrightarrow{a}_{\mathsf{p}} x' \parallel y}$$

$$\frac{x \xrightarrow{a}_{\mathsf{p}} \surd,\ y \xrightarrow{b}_{\mathsf{p}} \surd}{x \mid y \xrightarrow{c}_{\mathsf{p}} \surd}\ a \mid b = c \quad \frac{x \xrightarrow{a}_{\mathsf{p}} \surd,\ y \xrightarrow{b}_{\mathsf{p}} y'}{x \mid y \xrightarrow{c}_{\mathsf{p}} y'}\ a \mid b = c$$

$$\frac{x \xrightarrow{a}_{\mathsf{p}} x',\ y \xrightarrow{b}_{\mathsf{p}} \surd}{x \mid y \xrightarrow{c}_{\mathsf{p}} x'}\ a \mid b = c \quad \frac{x \xrightarrow{a}_{\mathsf{p}} x',\ y \xrightarrow{b}_{\mathsf{p}} y'}{x \mid y \xrightarrow{c}_{\mathsf{p}} x' \parallel y'}\ a \mid b = c$$

$$\frac{x \xrightarrow{a}_{\mathsf{p}} \surd}{\partial_H(x) \xrightarrow{a}_{\mathsf{p}} \surd}\ a \notin H \quad \frac{x \xrightarrow{a}_{\mathsf{p}} x'}{\partial_H(x) \xrightarrow{a}_{\mathsf{p}} \partial_H(x')}\ a \notin H$$

$$\frac{\langle t_X | E\rangle \xrightarrow{a}_{\mathsf{p}} \surd}{\langle X | E\rangle \xrightarrow{a}_{\mathsf{p}} \surd}\ X = t_X \in E \quad \frac{\langle t_X | E\rangle \xrightarrow{a}_{\mathsf{p}} x'}{\langle X | E\rangle \xrightarrow{a}_{\mathsf{p}} x'}\ X = t_X \in E$$

Proof. Straightforward, by induction on the structure of I. □

A *bisimulation* B is a triple of symmetric binary relations $B_{\mathbf{P}} \subseteq \mathcal{T}_{\mathbf{P}} \times \mathcal{T}_{\mathbf{P}}$, $B_{\mathbf{I}} \subseteq \mathcal{T}_{\mathbf{I}} \times \mathcal{T}_{\mathbf{I}}$, and $B_{\mathbf{C}} \subseteq \mathcal{T}_{\mathbf{C}} \times \mathcal{T}_{\mathbf{C}}$ such that:

- if $B_{\mathbf{P}}(P_1, P_2)$ and $P_1 \xrightarrow{a}_{\mathsf{p}} \surd$, then $P_2 \xrightarrow{a}_{\mathsf{p}} \surd$;
- if $B_{\mathbf{P}}(P_1, P_2)$ and $P_1 \xrightarrow{a}_{\mathsf{p}} P_1'$, then there exists a $P_2' \in \mathcal{T}_{\mathbf{P}}$ such that $P_2 \xrightarrow{a}_{\mathsf{p}} P_2'$ and $B_{\mathbf{P}}(P_1', P_2')$;
- if $B_{\mathbf{I}}(I_1, I_2)$ and $f.m@g \sqsubseteq^{N_1} I_1$, then there exists an $N_2 \in \mathcal{T}_{\mathbf{Z}}^{\mathrm{INT}}$ such that $f.m@g \sqsubseteq^{N_2} I_2$ and $N_1 = N_2$;
- if $B_{\mathbf{C}}(C_1, C_2)$ and C_1 hasIF I_1, then there exists an $I_2 \in \mathcal{T}_{\mathbf{I}}$ such that C_2 hasIF I_2 and $B_{\mathbf{I}}(I_1, I_2)$;
- if $B_{\mathbf{C}}(C_1, C_2)$ and $C_1 \xrightarrow{a}_{\mathsf{c}} \surd$, then $C_2 \xrightarrow{a}_{\mathsf{c}} \surd$;
- if $B_{\mathbf{C}}(C_1, C_2)$ and $C_1 \xrightarrow{a}_{\mathsf{c}} C_1'$, then there exists a $C_2' \in \mathcal{T}_{\mathbf{C}}$ such that $C_2 \xrightarrow{a}_{\mathsf{c}} C_2'$ and $B_{\mathbf{C}}(C_1', C_2')$.

Table 7. Additional rules for operational semantics of ACC+REC

$$\frac{}{f.m@g \sqsubseteq^1 f.m@g} \qquad \frac{}{f.m@g \sqsubseteq^0 f'.m'@g'}\ f \neq f' \vee m \neq m' \vee g \neq g'$$

$$\frac{}{f.m@g \sqsubseteq^{-1} {\sim}g.m@f} \qquad \frac{}{f.m@g \sqsubseteq^0 {\sim}g'.m'@f'}\ f \neq f' \vee m \neq m' \vee g \neq g'$$

$$\frac{}{f.m@g \sqsubseteq^0 0} \qquad \frac{f.m@g \sqsubseteq^k i}{f.m@g \sqsubseteq^{-k} -i} \qquad \frac{f.m@g \sqsubseteq^k i,\ f.m@g \sqsubseteq^l j}{f.m@g \sqsubseteq^{k+l} i+j}$$

$$\frac{}{\mathsf{c}(i,x)\ \mathsf{hasIF}\ i} \qquad \frac{u\ \mathsf{hasIF}\ i,\ v\ \mathsf{hasIF}\ j}{u \parallel v\ \mathsf{hasIF}\ i+j}$$

$$\frac{f.m@g \sqsubseteq^k i,\ \mathsf{sg}(k)=1}{f.m@g \sqsubseteq^+ i} \qquad \frac{f.m@g \sqsubseteq^k i,\ \mathsf{sg}(k)=-1}{f.m@g \sqsubseteq^- i}$$

$$\frac{u\ \mathsf{hasIF}\ i,\ f.m@g \sqsubseteq^+ i}{f.m@g \sqsubseteq^+ \mathsf{IF}(u)} \qquad \frac{u\ \mathsf{hasIF}\ i,\ f.m@g \sqsubseteq^- i}{f.m@g \sqsubseteq^- \mathsf{IF}(u)}$$

$$\frac{x \xrightarrow{f.m@g}_{\mathsf{p}} \surd,\ f.m@g \sqsubseteq^+ i}{\mathsf{c}(i,x) \xrightarrow{f.m@g}_{\mathsf{c}} \surd} \qquad \frac{x \xrightarrow{{\sim}f.m@g}_{\mathsf{p}} \surd,\ g.m@f \sqsubseteq^- i}{\mathsf{c}(i,x) \xrightarrow{{\sim}f.m@g}_{\mathsf{c}} \surd} \qquad \frac{x \xrightarrow{f.m\underline{@}g}_{\mathsf{p}} \surd}{\mathsf{c}(i,x) \xrightarrow{f.m\underline{@}g}_{\mathsf{c}} \surd}$$

$$\frac{x \xrightarrow{f.m@g}_{\mathsf{p}} x',\ f.m@g \sqsubseteq^+ i}{\mathsf{c}(i,x) \xrightarrow{f.m@g}_{\mathsf{c}} \mathsf{c}(i,x')} \qquad \frac{x \xrightarrow{{\sim}f.m@g}_{\mathsf{p}} x',\ g.m@f \sqsubseteq^- i}{\mathsf{c}(i,x) \xrightarrow{{\sim}f.m@g}_{\mathsf{c}} \mathsf{c}(i,x')} \qquad \frac{x \xrightarrow{f.m\underline{@}g}_{\mathsf{p}} x'}{\mathsf{c}(i,x) \xrightarrow{f.m\underline{@}g}_{\mathsf{c}} \mathsf{c}(i,x')}$$

$$\frac{u \xrightarrow{f.m@g}_{\mathsf{c}} \surd,\ f.m@g \sqsubseteq^+ \mathsf{IF}(u \parallel v)}{u \parallel v \xrightarrow{f.m@g}_{\mathsf{c}} v} \qquad \frac{u \xrightarrow{{\sim}f.m@g}_{\mathsf{c}} \surd,\ g.m@f \sqsubseteq^- \mathsf{IF}(u \parallel v)}{u \parallel v \xrightarrow{{\sim}f.m@g}_{\mathsf{c}} v} \qquad \frac{u \xrightarrow{f.m\underline{@}g}_{\mathsf{c}} \surd}{u \parallel v \xrightarrow{f.m\underline{@}g}_{\mathsf{c}} v}$$

$$\frac{v \xrightarrow{f.m@g}_{\mathsf{c}} \surd,\ f.m@g \sqsubseteq^+ \mathsf{IF}(u \parallel v)}{u \parallel v \xrightarrow{f.m@g}_{\mathsf{c}} u} \qquad \frac{v \xrightarrow{{\sim}f.m@g}_{\mathsf{c}} \surd,\ g.m@f \sqsubseteq^- \mathsf{IF}(u \parallel v)}{u \parallel v \xrightarrow{{\sim}f.m@g}_{\mathsf{c}} u} \qquad \frac{v \xrightarrow{f.m\underline{@}g}_{\mathsf{c}} \surd}{u \parallel v \xrightarrow{f.m\underline{@}g}_{\mathsf{c}} u}$$

$$\frac{u \xrightarrow{f.m@g}_{\mathsf{c}} u',\ f.m@g \sqsubseteq^+ \mathsf{IF}(u \parallel v)}{u \parallel v \xrightarrow{f.m@g}_{\mathsf{c}} u' \parallel v} \qquad \frac{u \xrightarrow{{\sim}f.m@g}_{\mathsf{c}} u',\ g.m@f \sqsubseteq^- \mathsf{IF}(u \parallel v)}{u \parallel v \xrightarrow{{\sim}f.m@g}_{\mathsf{c}} u' \parallel v} \qquad \frac{u \xrightarrow{f.m\underline{@}g}_{\mathsf{c}} u'}{u \parallel v \xrightarrow{f.m\underline{@}g}_{\mathsf{c}} u' \parallel v}$$

$$\frac{v \xrightarrow{f.m@g}_{\mathsf{c}} v',\ f.m@g \sqsubseteq^+ \mathsf{IF}(u \parallel v)}{u \parallel v \xrightarrow{f.m@g}_{\mathsf{c}} u \parallel v'} \qquad \frac{v \xrightarrow{{\sim}f.m@g}_{\mathsf{c}} v',\ g.m@f \sqsubseteq^- \mathsf{IF}(u \parallel v)}{u \parallel v \xrightarrow{{\sim}f.m@g}_{\mathsf{c}} u \parallel v'} \qquad \frac{v \xrightarrow{f.m\underline{@}g}_{\mathsf{c}} v'}{u \parallel v \xrightarrow{f.m\underline{@}g}_{\mathsf{c}} u \parallel v'}$$

$$\frac{u \xrightarrow{a}_{\mathsf{c}} \surd,\ v \xrightarrow{b}_{\mathsf{c}} \surd}{u \parallel v \xrightarrow{c}_{\mathsf{c}} \surd}\ a \mid b = c \qquad \frac{u \xrightarrow{a}_{\mathsf{c}} \surd,\ v \xrightarrow{b}_{\mathsf{c}} v'}{u \parallel v \xrightarrow{c}_{\mathsf{c}} v'}\ a \mid b = c$$

$$\frac{u \xrightarrow{a}_{\mathsf{c}} u',\ v \xrightarrow{b}_{\mathsf{c}} \surd}{u \parallel v \xrightarrow{c}_{\mathsf{c}} u'}\ a \mid b = c \qquad \frac{u \xrightarrow{a}_{\mathsf{c}} u',\ v \xrightarrow{b}_{\mathsf{c}} v'}{u \parallel v \xrightarrow{c}_{\mathsf{c}} u' \parallel v'}\ a \mid b = c$$

$$\frac{x \xrightarrow{f.m@g}_{\mathsf{p}} \surd,\ f.m@g \sqsubseteq^+ i}{\overline{\partial}_i(x) \xrightarrow{f.m@g}_{\mathsf{p}} \surd} \qquad \frac{x \xrightarrow{{\sim}f.m@g}_{\mathsf{p}} \surd,\ g.m@f \sqsubseteq^- i}{\overline{\partial}_i(x) \xrightarrow{{\sim}f.m@g}_{\mathsf{p}} \surd} \qquad \frac{x \xrightarrow{f.m\underline{@}g}_{\mathsf{p}} \surd}{\overline{\partial}_i(x) \xrightarrow{f.m\underline{@}g}_{\mathsf{p}} \surd}$$

$$\frac{x \xrightarrow{f.m@g}_{\mathsf{p}} x',\ f.m@g \sqsubseteq^+ i}{\overline{\partial}_i(x) \xrightarrow{f.m@g}_{\mathsf{p}} \overline{\partial}_i(x')} \qquad \frac{x \xrightarrow{{\sim}f.m@g}_{\mathsf{p}} x',\ g.m@f \sqsubseteq^- i}{\overline{\partial}_i(x) \xrightarrow{{\sim}f.m@g}_{\mathsf{p}} \overline{\partial}_i(x')} \qquad \frac{x \xrightarrow{f.m\underline{@}g}_{\mathsf{p}} x'}{\overline{\partial}_i(x) \xrightarrow{f.m\underline{@}g}_{\mathsf{p}} \overline{\partial}_i(x')}$$

Let $S \in \{\mathbf{P}, \mathbf{I}, \mathbf{C}\}$, and let $t_1, t_2 \in \mathcal{T}_S$. Then t_1 and t_2 are *bisimilar*, written $t_1 \leftrightarroweq t_2$, if there exists a bisimulation B such that $B_S(t_1, t_2)$.

The following congruence property of bisimilarity will be used in Section 10 to construct a model of ACC+REC.

Theorem 1 (Congruence). *Bisimilarity is a congruence with respect to the operators of* ACC+REC *to build terms of sort* $\mathbf{P}$, $\mathbf{I}$ *or* $\mathbf{C}$.

Proof. In the terminology of [12], $\mathbf{Z}$ is a given sort and the relations $f.m@g \sqsubseteq^N$, one for each $N \in \mathcal{T}_{\mathbf{Z}}^{\mathrm{INT}}$, constitute a relation parametrized by closed terms of the sort $\mathbf{Z}$. Because $\mathbf{Z}$ is a given sort, we can safely identify closed terms of sort $\mathbf{Z}$ that are semantically equivalent and replace the third property of bisimulations given above to:

- if $B_{\mathbf{I}}(I_1, I_2)$ and $f.m@g \sqsubseteq^N I_1$, then $f.m@g \sqsubseteq^N I_2$.

Because the relations $f.m@g \sqsubseteq^N$ constitute a relation parametrized by closed terms of a given sort, we can safely replace the rules for the operational semantics with the conclusions $f.m@g \sqsubseteq^+ i$ and $f.m@g \sqsubseteq^- i$ by the rules

$$\frac{f.m@g \sqsubseteq^N i}{f.m@g \sqsubseteq^+ i}\ \mathsf{sg}(N) = 1 \quad \text{and} \quad \frac{f.m@g \sqsubseteq^N i}{f.m@g \sqsubseteq^- i}\ \mathsf{sg}(N) = -1\ ,$$

where N stands for an arbitrary closed term from $\mathcal{T}_{\mathbf{Z}}^{\mathrm{INT}}$. By these replacements, bisimilarity becomes an instance of bisimilarity by the definition given in [12] and the rules for the operational semantics of ACC+REC become a complete transition system specification in panth format by the definitions given in [12]. Hence, it follows by Theorem 4 from [12] that bisimilarity is a congruence with respect to all operators of ACC+REC to build terms of sort $\mathbf{P}$, $\mathbf{I}$ or $\mathbf{C}$. □

10 A Bisimulation Model of ACC+REC

In this section, we construct a model of ACC+REC using the notion of bisimilarity defined in Section 9. It is a model in which all processes are finitely branching, i.e. they have at any stage only finitely many alternatives to proceed.

Henceforth, we will write $\mathfrak{I}_{\mathrm{INT}}$ for the initial model of INT, and $\mathbb{Z}$ for the set associated with the sort $\mathbf{Z}$ in $\mathfrak{I}_{\mathrm{INT}}$.

The *bisimulation model* $\mathfrak{B}_{\mathrm{ACC+REC}}$ is the expansion of $\mathfrak{I}_{\mathrm{INT}}$, the initial model of INT, with

- for each sort $S \in \{\mathbf{P}, \mathbf{I}, \mathbf{C}\}$, the set $\mathcal{T}_S/\underline{\leftrightarrow}$;
- for each constant $\Diamond_0 : S$ of ACC+REC with $S \in \{\mathbf{P}, \mathbf{I}, \mathbf{C}\}$, the element $\underline{\Diamond}_0 \in \mathcal{T}_S/\underline{\leftrightarrow}$ defined by $\underline{\Diamond}_0 = [\Diamond_0]_{\underline{\leftrightarrow}}$;
- for each operator $\Diamond_1 : S \to S'$ of ACC+REC with $S, S' \in \{\mathbf{P}, \mathbf{I}, \mathbf{C}\}$, the operation $\underline{\Diamond}_1 : \mathcal{T}_S/\underline{\leftrightarrow} \to \mathcal{T}_{S'}/\underline{\leftrightarrow}$ defined by $\underline{\Diamond}_1([t]_{\underline{\leftrightarrow}}) = [\Diamond_1(t)]_{\underline{\leftrightarrow}}$;
- for each operator $\Diamond_2 : S \times S' \to S''$ of ACC+REC with $S, S', S'' \in \{\mathbf{P}, \mathbf{I}, \mathbf{C}\}$, the operation $\underline{\Diamond}_2 : \mathcal{T}_S/\underline{\leftrightarrow} \times \mathcal{T}_{S'}/\underline{\leftrightarrow} \to \mathcal{T}_{S''}/\underline{\leftrightarrow}$ defined by $\underline{\Diamond}_2([t_1]_{\underline{\leftrightarrow}}, [t_2]_{\underline{\leftrightarrow}}) = [\Diamond_2(t_1, t_2)]_{\underline{\leftrightarrow}}$;

– for each operator $\#_{f.m@g} : \mathbf{I} \to \mathbf{Z}$ with $f, g \in \mathcal{L}$ and $m \in \mathcal{M}$, the operation $\underline{\#}_{f.m@g} : \mathcal{T}_{\mathbf{I}}/\underline{\leftrightarrows} \to \mathbb{Z}$ defined by $\underline{\#}_{f.m@g}([I]_{\underline{\leftrightarrows}})$ is the unique interpretation in $\mathfrak{I}_{\mathrm{INT}}$ of all $N \in \mathcal{T}_{\mathbf{Z}}^{\mathrm{INT}}$ for which $f.m@g \sqsubseteq^N I$.

The well-definedness of the operations associated with the operators of ACC+REC in $\mathfrak{B}_{\mathrm{ACC+REC}}$ follows immediately from Theorem 1, except for the operations associated with the operators $\#_{f.m@g}$. The well-definedness of the operations associated with the operators $\#_{f.m@g}$ in $\mathfrak{B}_{\mathrm{ACC+REC}}$ follows immediately from Lemma 1 and the definition of bisimilarity.

We have the following soundness result.

Theorem 2 (Soundness). *Let $S \in \{\mathbf{Z}, \mathbf{P}, \mathbf{I}, \mathbf{C}\}$ and let $t, t' \in \mathcal{T}_S$. Then $t = t'$ is derivable from the axioms of* ACC+REC *only if $t = t'$ holds in $\mathfrak{B}_{\mathrm{ACC+REC}}$.*

Proof. It is sufficient to prove the soundness of each axiom separately. Because $\mathfrak{B}_{\mathrm{ACC+REC}}$ is an expansion of $\mathfrak{I}_{\mathrm{INT}}$, it is not necessary to prove the soundness of the axioms of INT. For each of the remaining axioms except M1–M5, soundness is easily proved by constructing a witnessing bisimulation (for the witnessing bisimulations for the axioms of ACP+REC, see e.g. [1]). What remains are the proofs for axioms M1–M5. The soundness of these axioms follow immediately from the definition of $\underline{\#}_{f.m@g}$ and the rules of the operational semantics. □

11 Localized Processes

If processes are looked at in isolation, it is convenient to abstract from the loci at which they reside. This brings us to consider processes made up of actions of the forms $f.m$ and $\sim f.m$. These processes are called localized processes. In this section, we extend ACC with localized processes. The resulting theory is called $\mathrm{ACC}_{\mathrm{lp}}$.

Henceforth, actions from $\mathcal{A}$ will also be called non-localized actions, and processes made up of actions from $\mathcal{A}$ will also be called non-localized processes.

In $\mathrm{ACC}_{\mathrm{lp}}$, we have, in addition to the set $\mathcal{A}$ of non-localized actions, the set $\mathcal{LA}$ of *localized actions* consisting of:

– for each $f \in \mathcal{L}$ and $m \in \mathcal{M}$, the *active localized action* $f.m$;
– for each $f \in \mathcal{L}$ and $m \in \mathcal{M}$, the *passive localized action* $\sim f.m$.

Intuitively, these localized actions can be explained as follows:

– $f.m$ is the action by which a localized process requests a process residing at locus f to carry out method m;
– $\sim f.m$ is the action by which a localized process grants a request of a process residing at locus f to carry out method m.

It is not possible to perform localized actions synchronously.

Different from ACC, $\mathrm{ACC}_{\mathrm{lp}}$ has two sorts of processes. That is, $\mathrm{ACC}_{\mathrm{lp}}$ has the sorts $\mathbf{C}$, $\mathbf{P}$, $\mathbf{I}$ and $\mathbf{Z}$ from ACC, and in addition the sort $\mathbf{LP}$ of *localized*

Table 8. Axioms for placement of localized processes

$@_f(\delta) = \delta$	P1
$@_f(g.m) = g.m@f$	P2
$@_f(\sim g.m) = \sim g.m@f$	P3
$@_f(r + s) = @_f(r) + @_f(s)$	P4
$@_f(r \cdot s) = @_f(r) \cdot @_f(s)$	P5

processes. To build terms of sort **C**, $\mathrm{ACC}_{\mathrm{lp}}$ has the constants and operators of ACC to build terms of sort **C**. To build terms of sort **P**, $\mathrm{ACC}_{\mathrm{lp}}$ has the constants and operators of ACC to build terms of sort **P** and in addition the following operators:

– for each $f \in \mathcal{L}$, the unary *placement* operator $@_f : \mathbf{LP} \to \mathbf{P}$.

To build terms of sort **LP**, $\mathrm{ACC}_{\mathrm{lp}}$ has the following constants and operators:

– the *deadlock* constant $\delta : \mathbf{LP}$;
– for each $a \in \mathcal{LA}$, the *localized action* constant $a : \mathbf{LP}$;
– the binary *alternative composition* operator $+ : \mathbf{LP} \times \mathbf{LP} \to \mathbf{LP}$;
– the binary *sequential composition* operator $\cdot : \mathbf{LP} \times \mathbf{LP} \to \mathbf{LP}$;
– the binary *parallel composition* operator $\parallel : \mathbf{LP} \times \mathbf{LP} \to \mathbf{LP}$;
– the binary *left merge* operator $\lfloor\!\lfloor : \mathbf{LP} \times \mathbf{LP} \to \mathbf{LP}$;
– for each $H \subseteq \mathcal{A}$, the unary *encapsulation* operator $\partial_H : \mathbf{LP} \to \mathbf{LP}$.

To build terms of sort **I**, $\mathrm{ACC}_{\mathrm{lp}}$ has the constants and operators of ACC to build terms of sort **I**. To build terms of sort **Z**, $\mathrm{ACC}_{\mathrm{lp}}$ has the constants and operators of ACC to build terms of sort **Z**.

Terms of the different sorts are built as usual for a many-sorted signature. We assume that there are infinitely many variables of sort **LP**, including r, s, r' and s'.

The constants and operators to build terms of sort **LP** need no further explanation. They differ from the constants and operators to build terms of sort **P** in that: (i) the (non-localized) action constants are replaced by the localized action constants and (ii) the communication merge operator $|$ is removed.

Let L be a closed term of sort **LP**. Intuitively, the operators $@_f$ can be explained as follows:

– $@_f(L)$ behaves as L with each action $g.m$ replaced by $g.m@f$ and each action $\sim g.m$ replaced by $\sim g.m@f$.

In other words, $@_f$ turns localized processes into non-localized processes by placing them as a whole in locus f.

The axioms of $\mathrm{ACC}_{\mathrm{lp}}$ are the axioms of ACC, the axioms given in Tables 8 and 9, and copies of axioms A1–A7, CM2–CM4 and D1–D4 from Table 1 with x, y and z replaced by different variables of sort **LP**, a standing for an arbitrary

Table 9. Axiom for parallel composition of localized processes

$$\frac{}{r \parallel s = r \between s + s \between r} \quad \text{M1}$$

Table 10. Additional rules for operational semantics of $\mathrm{ACC}_{\mathrm{lp}}$

$$\frac{r \xrightarrow{g.m}_{\mathsf{lp}} \surd}{@_f(r) \xrightarrow{g.m@f}_{\mathsf{p}} \surd} \qquad \frac{r \xrightarrow{\sim g.m}_{\mathsf{lp}} \surd}{@_f(r) \xrightarrow{\sim g.m@f}_{\mathsf{p}} \surd} \qquad \frac{r \xrightarrow{g.m}_{\mathsf{lp}} r'}{@_f(r) \xrightarrow{g.m@f}_{\mathsf{p}} @_f(r')} \qquad \frac{r \xrightarrow{\sim g.m}_{\mathsf{lp}} r'}{@_f(r) \xrightarrow{\sim g.m@f}_{\mathsf{p}} @_f(r')}$$

constant of sort **LP** and H standing for an arbitrary subset of $\mathcal{LA}$. Axioms P1–P5 are the defining axioms of $@_f$. Axiom M1 replaces axiom CM1. The latter axiom is not suited for the localized case because it is not possible to perform localized actions synchronously.

Guarded recursion can be added to $\mathrm{ACC}_{\mathrm{lp}}$ as it is added to ACP in Section 3. We write $\mathrm{ACC}_{\mathrm{lp}}$+REC for $\mathrm{ACC}_{\mathrm{lp}}$ extended with the constants standing for the unique solutions of guarded recursive specifications and the axioms RDP and RSP.

In the structural operational semantics of $\mathrm{ACC}_{\mathrm{lp}}$+REC, the following relations are used in addition to the ones used in the structural operational semantics of ACC+REC:

- a unary relation $\xrightarrow{a}_{\mathsf{lp}} \surd \subseteq \mathcal{T}_{\mathbf{LP}}$, for each $a \in \mathcal{LA}$;
- a binary relation $\xrightarrow{a}_{\mathsf{lp}} \subseteq \mathcal{T}_{\mathbf{LP}} \times \mathcal{T}_{\mathbf{LP}}$, for each $a \in \mathcal{LA}$.

We write $L \xrightarrow{a}_{\mathsf{lp}} \surd$ instead of $L \in \xrightarrow{a}_{\mathsf{lp}} \surd$ and $L \xrightarrow{a}_{\mathsf{lp}} L'$ instead of $(L, L') \in \xrightarrow{a}_{\mathsf{lp}}$. The relations can be explained as follows:

- $L \xrightarrow{a}_{\mathsf{lp}} \surd$: localized process L is capable of first performing a and then terminating successfully;
- $L \xrightarrow{a}_{\mathsf{p}} L'$: localized process P is capable of first performing a and then proceeding as localized process P'.

The structural operational semantics of $\mathrm{ACC}_{\mathrm{lp}}$+REC is described by the rules for the operational semantics of ACC+REC, the rules given in Table 10, and copies of the rules without the side-condition $a \mid b = c$ from Table 6 with $\xrightarrow{a}_{\mathsf{p}} \surd$ and $\xrightarrow{a}_{\mathsf{p}}$ replaced by $\xrightarrow{a}_{\mathsf{lp}} \surd$ and $\xrightarrow{a}_{\mathsf{lp}}$, respectively, x, x', y and y' replaced by different variables of sort **LP**, a standing for an arbitrary constant of sort **LP** and H standing for an arbitrary subset of $\mathcal{LA}$.

Constructing a bisimulation model of $\mathrm{ACC}_{\mathrm{lp}}$+REC can be done on the same lines as constructing a bisimulation model of ACC+REC.

12 Conclusions

In this paper, we have built on earlier work on ACP and earlier work on interface groups. ACP was first presented in [4] and interface groups were first proposed

in [7]. We have introduced an interface group for process components and have presented a theory about process components of which that interface group forms part. The presented theory is a development on top of ACP. We have illustrated the use of the theory by means of an example, and have given a bisimulation semantics for process components which justifies the axioms of the theory.

Two interesting properties of the interface group for process components introduced in this paper are: (i) the interface combination operator $+$ is not idempotent and (ii) for each $f, g \in \mathcal{L}$ and $m \in \mathcal{M}$, the interface element constants $f.m@g$ and $\sim g.m@f$ are each other inverses. Property (i) allows for expressing that a process component expects from a number of process components an ability or promises a number of process components an ability. Property (ii) allows for establishing whether a system composed of a collection of process components is a closed system.

Like in [7], the inclusion of behavioural information in component interfaces has been deliberately rejected in order to have orthogonality between component interfaces and component behaviours. The distinction between active interface elements and passive interface elements made in this paper corresponds to the distinction between import services and export services made in [13]. Adaptations of module algebra [3] that allow for this kind of distinction are investigated in [8]. However, interface groups are not considered in those investigations.

Processes as considered in ACP have been combined with interfaces before in the tool-supported formalisms for description and analysis of processes with data known as μCRL [10] and PSF [11]. However, in μCRL and PSF, interfaces serve for determining whether descriptions of processes are well-formed only.

References

1. J. C. M. Baeten and C. Verhoef. Concrete process algebra. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume IV, pages 149–268. Oxford University Press, Oxford, 1995.
2. J. C. M. Baeten and W. P. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, 1990.
3. J. A. Bergstra, J. Heering, and P. Klint. Module algebra. *Journal of the ACM*, 37(2):335–372, 1990.
4. J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1/3):109–137, 1984.
5. J. A. Bergstra and J. W. Klop. Process algebra: Specification and verification in bisimulation semantics. In M. Hazewinkel, J. K. Lenstra, and L. G. L. T. Meertens, editors, *Proceedings Mathematics and Computer Science II*, volume 4 of *CWI Monograph*, pages 61–94. North-Holland, 1986.
6. J. A. Bergstra and C. A. Middelburg. Thread algebra with multi-level strategies. *Fundamenta Informaticae*, 71(2/3):153–182, 2006.
7. J. A. Bergstra and A. Ponse. Interface groups for analytic execution architectures. Electronic Report PRG0601, Programming Research Group, University of Amsterdam, 2006.

8. L. M. G. Feijs and Y. Qian. Component algebra. *Science of Computer Programming*, 42:173–228, 2002.
9. W. J. Fokkink. *Introduction to Process Algebra*. Texts in Theoretical Computer Science, An EATCS Series. Springer-Verlag, Berlin, 2000.
10. J. F. Groote and A. Ponse. The syntax and semantics of μCRL. In A. Ponse, C. Verhoef, and S. F. M. van Vlijmen, editors, *Algebra of Communicating Processes 1994*, Workshops in Computing Series, pages 26–62. Springer-Verlag, 1995.
11. S. Mauw and G. J. Veltink. A process specification formalism. *Fundamenta Informaticae*, 13(2):85–139, 1990.
12. C. A. Middelburg. An alternative formulation of operational conservativity with binding terms. *Journal of Logic and Algebraic Programming*, 55(1/2):1–19, 2003.
13. C. Pahl. An ontology for software component matching. In M. Pezzè, editor, *FASE 2003*, volume 2621 of *Lecture Notes in Computer Science*, pages 6–21. Springer-Verlag, 2003.
14. D. Sannella and A. Tarlecki. Algebraic preliminaries. In E. Astesiano, H.-J. Kreowski, and B. Krieg-Brückner, editors, *Algebraic Foundations of Systems Specification*, pages 13–30. Springer-Verlag, Berlin, 1999.
15. M. Wirsing. Algebraic specification. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 675–788. Elsevier, Amsterdam, 1990.

Within this series the following reports appeared.

[PRG0706] J.A. Bergstra, Y. Hirschfeld, and J.V. Tucker, *Skew Meadows*, Programming Research Group - University of Amsterdam, 2007.

[PRG0705] J.A. Bergstra, Y. Hirschfeld, and J.V. Tucker, *Meadows*, Programming Research Group - University of Amsterdam, 2007.

[PRG0704] J.A. Bergstra and C.A. Middelburg, *Machine Structure Oriented Control Code Logic (Extended Version)*, Programming Research Group - University of Amsterdam, 2007.

[PRG0703] J.A. Bergstra and C.A. Middelburg, *On the Operating Unit Size of Load/Store Architectures*, Programming Research Group - University of Amsterdam, 2007.

[PRG0702] J.A. Bergstra and A. Ponse, *Interface Groups and Financial Transfer Architectures*, Programming Research Group - University of Amsterdam, 2007.

[PRG0701] J.A. Bergstra, I. Bethke, and M. Burgess, *A Process Algebra Based Framework for Promise Theory*, Programming Research Group - University of Amsterdam, 2007.

[PRG0610] J.A. Bergstra and C.A. Middelburg, *Parallel Processes with Implicit Computational Capital*, Programming Research Group - University of Amsterdam, 2006.

[PRG0609] B. Diertens, *Software (Re-)Engineering with PSF II: from architecture to implementation*, Programming Research Group - University of Amsterdam, 2006.

[PRG0608] A. Ponse and M.B. van der Zwaag, *Risk Assessment for One-Counter Threads*, Programming Research Group - University of Amsterdam, 2006.

[PRG0607] J.A. Bergstra and C.A. Middelburg, *Synchronous Cooperation for Explicit Multi-Threading*, Programming Research Group - University of Amsterdam, 2006.

[PRG0606] J.A. Bergstra and M. Burgess, *Local and Global Trust Based on the Concept of Promises*, Programming Research Group - University of Amsterdam, 2006.

[PRG0605] J.A. Bergstra and J.V. Tucker, *Division Safe Calculation in Totalised Fields*, Programming Research Group - University of Amsterdam, 2006.

[PRG0604] J.A. Bergstra and A. Ponse, *Projection Semantics for Rigid Loops*, Programming Research Group - University of Amsterdam, 2006.

[PRG0603] J.A. Bergstra and I. Bethke, *Predictable and Reliable Program Code: Virtual Machine-based Projection Semantics (submitted for inclusion in the Handbook of Network and Systems Administration)*, Programming Research Group - University of Amsterdam, 2006.

[PRG0602] J.A. Bergstra and A. Ponse, *Program Algebra with Repeat Instruction*, Programming Research Group - University of Amsterdam, 2006.

[PRG0601] J.A. Bergstra and A. Ponse, *Interface Groups for Analytic Execution Architectures*, Programming Research Group - University of Amsterdam, 2006.

[PRG0505] B. Diertens, *Software (Re-)Engineering with PSF*, Programming Research Group - University of Amsterdam, 2005.

[PRG0504] P.H. Rodenburg, *Piecewise Initial Algebra Semantics*, Programming Research Group - University of Amsterdam, 2005.

[PRG0503] T.D. Vu, *Metric Denotational Semantics for BPPA*, Programming Research Group - University of Amsterdam, 2005.

[PRG0502] J.A. Bergstra, I. Bethke, and A. Ponse, *Decision Problems for Pushdown Threads*, Programming Research Group - University of Amsterdam, 2005.

[PRG0501] J.A. Bergstra and A. Ponse, *A Bypass of Cohen's Impossibility Result*, Programming Research Group - University of Amsterdam, 2005.

[PRG0405] J.A. Bergstra and I. Bethke, *An Upper Bound for the Equational Specification of Finite State Services*, Programming Research Group - University of Amsterdam, 2004.

[PRG0404] J.A. Bergstra and C.A. Middelburg, *Thread Algebra for Strategic Interleaving*, Programming Research Group - University of Amsterdam, 2004.

[PRG0403] B. Diertens, *A Compiler-projection from PGLEc.MSPio to Parrot*, Programming Research Group - University of Amsterdam, 2004.

[PRG0402] J.A. Bergstra and I. Bethke, *Linear Projective Program Syntax*, Programming Research Group - University of Amsterdam, 2004.

[PRG0401] B. Diertens, *Molecular Scripting Primitives*, Programming Research Group - University of Amsterdam, 2004.

[PRG0302] B. Diertens, *A Toolset for PGA*, Programming Research Group - University of Amsterdam, 2003.

[PRG0301] J.A. Bergstra and P. Walters, *Projection Semantics for Multi-File Programs*, Programming Research Group - University of Amsterdam, 2003.

[PRG0201] I. Bethke and P. Walters, *Molecule-oriented Java Programs for Cyclic Sequences*, Programming Research Group - University of Amsterdam, 2002.

The above reports and more are available through the website: www.science.uva.nl/research/prog/

Electronic Report Series

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ Amsterdam
the Netherlands

www.science.uva.nl/research/prog/