On Calculi for Context-Aware Coordination

Pietro Braione and Gian Pietro Picco

Dipartimento di Elettronica e Informazione, Politecnico di Milano
P.za Leonardo da Vinci, 32, I-20133 Milano, Italy
[braione,picco]@elet.polimi.it

Abstract. Modern distributed computing demands unprecedented levels of dynamicity and reconfiguration. Mobile computing, peer-to-peer networks, computational grids, multiagent systems, are examples of domains exhibiting a continuously changing system configuration. In these settings, the *context* where computation occurs is not only dynamically changing, but also affecting the components' behavior in a fundamental way, by enabling or inhibiting some of their actions.
In this paper we are concerned with formal specification. Process calculi are a common choice for specifying concurrent and distributed systems. Unfortunately, context representation is rarely addressed explicitly. A few approaches provide limited expressiveness, in that they force the specifier to exploit a predefined and partial notion of context.
This paper is a first step in laying the formal foundation for a process calculi specification style that: *i)* fosters a coordination approach by sharply separating the process behavior from computational context defined by system changes; *ii)* enables the specifier to define her notion of context and the rules governing how it affects the application process behavior.

1 Introduction

Modern distributed computing demands unprecedented levels of dynamicity and reconfiguration. Mobile computing scenarios modify the physical topology of the system, and make communication transient and opportunistic. Mobile code changes the software fabric of a distributed computing system into a fluid one, by allowing program fragments to travel across hosts. Multiagent systems, computational grids, and peer-to-peer networks are other examples of domains where the physical or logical structure of the system is continuously under reconfiguration.

The hallmark of these scenarios is that the *context* where computation occurs is no longer fixed as in traditional distributed computing. A mobile host often accesses only services that are provided by hosts in range. Mobile agents may use only the resources available on the host they reside on. Queries in a peer-to-peer system return results that depend on the current set of connected peers. Interestingly, context not only defines the allowed scope for interaction, but also affects the components' behavior in a fundamental way, by enabling or inhibiting some of their actions. A message can be reported by a soldier to the commander of a patrol only if the communication medium ensures an adequate level of secrecy. The result of a computation carried by a swarm of mobile agents can be

computed only when they are all co-located on the same host. A particular music file can be downloaded from a peer only if no closer one offers an equivalent file.

Dealing with a changing computational context is the fundamental challenge of mobility and, in general, modern distributed computing. As noted in [1], a coordination perspective is helpful in addressing this challenge, in that it allows to separate sharply the details of a component's internal behavior, concerned with application issues, from the details of the surrounding computational context, represented through coordination abstractions. Essentially, a coordination perspective allows context to emerge as a first-class element, and be treated accordingly. However, the complexity of the issues involved requires the use of formal thinking when tackling the design of systems in this environment.

Process calculi are a common way to formally describe concurrent, distributed systems, and have recently been used successfully for specifying the semantics of coordination models and languages. Unfortunately, these calculi only rarely address directly (i.e., with appropriate abstractions) the modeling of a changing computational context. Moreover, in these few cases the specifier is constrained by a (rather rigid) notion of context built-in the calculus. For instance, the Ambient calculus [2] assumes space organized in hierarchical locations, and the context perceived by a process is determined by its position in the space tree. The resulting approach is undoubtedly elegant, in that the space structure is naturally mirrored by the very syntactic structure of terms. On the other hand, it forces *a priori* a space structure that may not be suitable for the specification task at hand. As an example, the inherently graph-based nature of mobile ad hoc networks and peer-to-peer systems can hardly be reduced to a hierarchy, and so are cases where physical or logical space areas (and hence contexts) overlap.

The specification of middleware and languages assuming alternative notions of space and context becomes then cumbersome—as we experienced directly. Our initial goal was in fact to formalize the LIME middleware [3] using process calculi. It was our intent that this formalization would allow us to evaluate this specification style against a real coordination middleware, and elicit the trade-offs against the state-based style of the original Mobile UNITY formalization of LIME [4]. We soon discovered that describing the LIME semantics using a "raw" process calculus leads to a cumbersome specification, precisely because contextual information gets buried in technical details. Proper abstractions are required to simplify both the development and the understanding of the specification.

The contribution of this paper is a first step in laying the formal foundation for a process calculi specification style that: *i)* fosters a coordination approach by sharply separating the process behavior from computational context defined by system changes; *ii)* enables the specifier to define her notion of context and the rules governing how it affects the application process behavior. Hence, decoupling and flexibility in representing context are our driving motivations, with the goal of obtaining a formalism expressive enough to be used to model a real middleware. Process calculi is the formal tool we use to shape our ideas.

Our approach is based on *reactive systems* (RS), a kind of process calculi inspired by Berry and Boudol's Chemical Abstract Machine (CHAM) [5]. In a RS

the elementary computational steps are described as transitions (the chemical reactions) causing the rewriting of terms representing the current configuration of the system, expressed as a multiset of components (the chemical solution). Any such transition is interpreted as an interaction between a number of components which come in contact. The difficulty of representing a dynamic, subjective behavior in RSs stems from the fact that interactions are not disciplined: every computational context may host any interaction.

In this work, we generalize the notion of RS into a new class that we call *contextual reactive systems* (CRS). In these systems, it is possible to specify the computational context under which a class of interactions is allowed, and the contexts under which it is not. This allows to inhibit some behaviors according to the features of the context under which they should have been performed.

Our treatment of CRSs relies on categories. Capturing the fundamental properties at this high level of abstraction allows to concentrate on the core concepts without being distracted by the technicalities of a specific calculus. Moreover, it enables the definition of families of concrete calculi sharing the same abstract nature. To make the presentation more concrete, however, we show how the categorical framework can be instantiated in a process calculus, and demonstrate its effectiveness by formalizing some fragments of the LIME middleware [3].

This paper is organized as follows. Section 2 introduces a categorical formalization of reactive systems, and a simple Linda-like calculus that serves the twofold purpose of illustrating how a reactive system can implemented by a calculus, and of serving as a reference example. Section 3 highlights the limitations of reactive systems when dealing with context. Section 4 contains the main contribution of this paper as a formalization of the notion of CRS. Section 5 shows how CRSs can be used in a process calculus and discusses the advantages of our approach. Section 6 provides an example by illustrating a fragment of a formalization of LIME. Section 7 elaborates on the findings of this paper, by suggesting relationships with other approaches, as well as avenues for future research. Finally, Section 8 ends the paper with some brief concluding remarks.

2 Reactive Systems

The definition of reactive system assumed in this paper is a category-theoretic one, slightly adapted from Leifer [6] and from Sassone and Sobociński [7].

Definition 1 (Reactive system). *A* reactive system (RS) *is a* $(\mathcal{C}, I, \mathbf{R}, \mathcal{D})$ *quadruple, where* $\mathcal{C}$ *is a category,* $I \in Ob\mathcal{C}$, $\mathbf{R} \subseteq \bigcup_{x \in Ob\,\mathcal{C}} \mathcal{C}(I,x) \times \mathcal{C}(I,x)$, *and* $\mathcal{D} \leq \mathcal{C}$ *is composition-reflecting, i.e.,* $D_0\, D_1 \in Mo\,\mathcal{D} \implies D_i \in Mo\,\mathcal{D}$, $i = 0, 1$.

We analyze each point of this definition. The morphisms in $\mathcal{C}$ are called the *contexts* of the system. Among them, the *ground contexts* denote processes. Ground contexts generalize the terms of a calculus, while all the other contexts generalize the "terms with a hole" obtained by replacing exactly one subterm of a calculus' term with a special symbol $-$, the *hole*. Replacing the hole with a compatible subterm is abstractly modeled, in the categorical framework, by

morphism composition: $C\,C'$ corresponds to the term $C[C'/-]$. The objects of $\mathcal{C}$ serve the purpose to disciplinate composition, just like a sorting discipline is used to disciplinate substitution in a term algebra. Ground contexts are distinguished in the categorical framework by imposing that their source object is I, i.e., each of them belongs to some hom-sets $\mathcal{C}(I, x)$. I is not necessarily initial.

$\mathbf{R}$ is the set of the *elementary rules*. An elementary rule is a pair (l, r) of ground contexts, where l is named the *redex* and r the *contractum* of the rule. Elementary rules express the basic interactions for some simple, paradigmatic configurations. They are extended to form *composite rules*. The extension is performed as in the λ-calculus: When a redex appears as a subterm of another term, *in some cases* it may be reduced to its contractum leaving the containing term (the context) unchanged. This idea is captured in the category-theoretic framework by defining the subcategory $\mathcal{D} \leq \mathcal{C}$ of *reactive contexts*, which specifies the contexts under which a rule may fire, and the following relationship:

Definition 2 (Reaction relationship). *The* reaction relationship, $\rightarrow$, *is defined as follows:*

$$a \rightarrow a' \quad \Longleftrightarrow \quad \exists\ (l, r) \in \mathbf{R},\ D \in Mo\,\mathcal{D}\ .\ a = Dl \wedge a' = Dr.$$

This relationship contains all the rules (both elementary and composite) of the RS, hence $\mathbf{R} \subseteq \rightarrow \subseteq \bigcup_{x \in \mathrm{Ob}\,\mathcal{C}} \mathcal{C}(I, x) \times \mathcal{C}(I, x)$. Restricting the context under which elementary rules can fire to a predefined class of "evaluation contexts" is a technique dating back at least to Plotkin [8], where it was used to impose evaluation disciplines over λ-calculi. In process calculi, reactive contexts are instead used to forbid the computation of "switched off" processes, e.g. processes under a prefix. Composition-reflectivity is necessary to ensure some basic properties to RSs.

Example. To make our formulation of RSs more concrete, we introduce a process calculus[1] describing a data-based coordination system inspired by Linda. Processes are sequential compositions of primitive actions. They execute concurrently and coordinate themselves by exchanging tuples through a global data space. The calculus' terms are generated by the abstract syntax in Table 1, where v is a value in the set $\mathbf{V} = \{\mathrm{v}_0, \mathrm{v}_1, \ldots\}$, c a coordination primitive (in or out), P a process, T a tuple, and C a configuration. Infinite behaviors can be represented by processes in the form $\mathsf{rec}X.P$, where X is a process variable and P a process containing one guarded occurrence of X. We also require that all the occurrences of process variables in a term generated by P are bound by some rec. We define structural congruence $\equiv$ as the smallest congruence on terms such that:

$$\begin{array}{ll} C_1 \mid C_2 \equiv C_2 \mid C_1 & (C_1 \mid C_2) \mid C_3 \equiv C_1 \mid (C_2 \mid C_3) \\ C \mid 0 \equiv C & \mathsf{rec}X.P \equiv P[\mathsf{rec}X.P/X] \end{array}$$

Structural congruence equates terms which differ syntactically and nevertheless denote a same configuration. The operational semantics of the calculus is presented in Table 1: the rules are self-explanatory, and are not commented.

[1] The calculus is a slight modification of the one found in [9].

Syntax:	Semantics:	
$C ::= P \mid T \mid C \,\vert\, C$	$\mathsf{in}(v).P \,\vert\, \langle v\rangle \to P$	(2.1)
$P ::= 0 \mid X \mid c.P \mid \mathsf{rec}X.P$	$\mathsf{out}(v).P \to P \,\vert\, \langle v\rangle$	(2.2)
$T ::= \langle v\rangle$	$\dfrac{C \to C'}{C \,\vert\, P \to C' \,\vert\, P}$	(2.3)
$c ::= \mathsf{in}(v) \mid \mathsf{out}(v)$	$\dfrac{C \to C'}{C \,\vert\, T \to C' \,\vert\, T}$	(2.4)
	$\dfrac{D \equiv C \;\; C \to C' \;\; C' \equiv D'}{D \to D'}$	(2.5)

Table 1. Syntax and semantics of the $\mathbf{L}_{\mathrm{RS}}$ system.

Let us now see how the specification of $\mathbf{L}_{\mathrm{RS}}$ maps onto our definition of reactive systems. Elementary rules are represented by the semantic rules (2.1) and (2.2) in Table 1. Contexts are all the congruence classes of "terms with (at most) a hole". Reactive contexts are all the congruence classes of terms in the form $- \,|\, C$. Composite rules are expressed by the semantic rules (2.3) and (2.4) in Table 1, by recursion over the subclass of reactive contexts containing all the (congruence classes of) terms having the form $- \,|\, P$ and $- \,|\, T$.

The categorical definition of RS evidences some of its relevant features. First, while a context denotes a *locus*, i.e., an incomplete entity able to host another entity, a reactive context denotes a *computational locus*, i.e., a locus able to host an entity with a behavior and let it compute. Second, rules can be applied orthogonally under reactive contexts, i.e., *any* reactive context may host *any* reaction—with a correct composition typing. Hence, a reactive context can neither limit nor extend the internal behaviors of the entities it hosts. Third, by the very definition of composite rule, a reactive context is unaffected by the internal behaviors of the entities it hosts. We can summarize the last two observations by saying that computations in a RS are *pure* with respect to their context. As we describe in the rest of the paper, this orthogonality between reaction and context is the most relevant obstacle towards the decoupling and expressiveness we demand, and is lifted by our definition of contextual reactive systems.

3 Motivation

In the computing scenarios we target, computation is affected by the context[2] surrounding it (e.g., communication parties in range, set of services available, mobile agents co-located on the same host), and such context is dynamically and

[2] It is worth noting how here we use the term *context* to refer to a concept that is different from the one defined for reactive systems.

continuously changing. In these scenarios, the interactions between the components of an application often depend on the current configuration of the context.

Unfortunately, in some cases these situations are not easily modeled by means of a RS. We illustrate the issue with a simple example. A shared tuple space is used by printers to advertise their presence, and by processes to send them their jobs. A high-level print primitive is available to processes, whose effect is to automatically direct the job to the best printer among those currently advertised. Assuming, for the sake of simplicity, that only low-quality raw printers and high-quality PostScript printers are available, a possible configuration is:

$$\langle \text{pr:ps} \rangle \mid \langle \text{pr:raw} \rangle \mid \mathsf{print}(\text{txt}).0,$$

which contains two advertisements and a process which prints a job and then terminates. Our goal is to specify the semantics of print using a RS. In the first place, we must formalize how print generates jobs for both low-quality and high-quality printers. This can be done by rules in the form:

$$\mathsf{print}(\text{txt}).P \mid \langle \text{pr:raw} \rangle \rightarrow P \mid \langle \text{job,txt,raw} \rangle$$
$$\mathsf{print}(\text{txt}).P \mid \langle \text{pr:ps} \rangle \rightarrow P \mid \langle \text{job,txt,ps} \rangle$$

But how can we ensure that print generates jobs only for the PostScript printer, in the presence of both printers? We should forbid redexes as $\mathsf{print}(\text{txt}).P \mid \langle \text{pr:raw} \rangle$ to fire when they are placed in the same configuration with a tuple $\langle \text{pr:ps} \rangle$. Nevertheless, if we consider all the "contexts with a tuple" $- \mid \langle \ldots \rangle$ as reactive, this cannot be achieved since RSs allow *any* interaction to occur unrestrained under *any* reactive context. In the previous example with one printing processes and two advertisements, the system may perform one of two different transitions:

$$\langle \text{pr:ps} \rangle \mid \langle \text{pr:raw} \rangle \mid \mathsf{print}(\text{txt}).0 \rightarrow \langle \text{pr:ps} \rangle \mid \langle \text{pr:raw} \rangle \mid 0 \mid \langle \text{job,txt,raw} \rangle$$
$$\langle \text{pr:ps} \rangle \mid \langle \text{pr:raw} \rangle \mid \mathsf{print}(\text{txt}).0 \rightarrow \langle \text{pr:ps} \rangle \mid \langle \text{pr:raw} \rangle \mid 0 \mid \langle \text{job,txt,ps} \rangle$$

which are obtained by composing respectively the first and the second elementary rule with the reactive contexts $- \mid \langle \text{pr:ps} \rangle$ and $- \mid \langle \text{pr:raw} \rangle$.

This simple example evidences a characteristic of RSs that is central to the theme of this paper: It is not possible to forbid the reduction of a redex based on the properties of the context it is immersed in.

Besides being a limiting factor towards the ability to express computations that depend on the configuration of the physical context, this kind of issue is frequently found in many aspects of coordination languages. For instance, similar problems arise when modeling the semantics of probe operations and reactive programming [9]. Moreover, they also arise in modeling the semantics of LIME, as pointed out in Section 6. We characterize the problem more precisely by considering an extension of $\mathbf{L}_{\text{RS}}$ with the bulk operation ing, which atomically removes all the tuples matching a specified value v. Removal of an unlimited number of matching tuples can be specified[3] through an infinite number of rules

[3] For the sake of simplicity we assume that, if no matching tuple is available, the process fails by becoming inert.

in the form $\mathsf{ing}(v).P \mid \prod_n \langle v \rangle \to P$. Nevertheless, we cannot ensure that ing always removes *all* the matching tuples in a configuration. In fact:

Claim. There is no extension of $\mathbf{L}_{\mathrm{RS}}$ capable of expressing the semantics of ing.

Proof (Sketch). Let us assume by contradiction that we can, i.e., that there is no rule $a \to a'$ where a $\mathsf{ing}(v)$ prefix is consumed and at least one $\langle v \rangle$ tuple is not. Let us define two functions ι and τ yielding respectively the number of processes in a with shape $\mathsf{ing}(v).P$, and the number of tuples in a with shape $\langle v \rangle$. (These functions can be defined by induction.) Then, if $\iota(a', v) = \iota(a, v) - 1$, it must be $\tau(a', v) = 0$. Let us consider any rule $a \to a'$ of the calculus, such as $\iota(a', v) = \iota(a, v) - 1$, and let us compose it with the reactive context $- \mid \langle v \rangle$. We obtain a composite rule $a'' \to a'''$ such as $\iota(a''', v) = \iota(a'', v) - 1$ and $\tau(a''', v) > 0$. This contradicts our assumption. □

4 Contextual Reactive Systems

In the previous section we pointed out that the impossibility of inhibiting some reductions of a RS based on the properties of the context surrounding the redex hampers the style of modeling coordination constructs we are targeting. In this section we will define a new class of RSs overcoming this issue.

4.1 Fundamentals

To understand the spirit of our solution, let us focus on extending $\mathbf{L}_{\mathrm{RS}}$ with ing, as discussed earlier. The problem we identified is that an elementary rule of ing is allowed to fire under any reactive context, including those containing additional matching tuples that hence ing cannot remove. In principle, one could solve the issue by simply removing from $\mathcal{D}$ all the reactive contexts containing one or more tuples. The obvious disadvantage of this solution is its lack of generality: all the other elementary rules, which in principle are not affected by the presence of tuples, must be modified accordingly. In essence, the whole calculus must be revised to add a single primitive. In this paper, we seek instead for a more flexible solution that limits the removal of the undesired contexts (i.e., those with matching tuples) only to the elementary rules for ing, and leaves the rest of the system unaffected.

In essence, the fundamental idea behind our approach is to lift the property of orthogonality between reactions and contexts characterizing RSs, and allow instead elementary rules to be extended only by *some* (i.e., not necessary by *any*) of the reactive contexts in $\mathcal{D}$. Composite rules can still be obtained by recursively extending a core set of elementary rules with a set reactive contexts, which belong to the part of the configuration left unaffected by the transition. The important difference with RSs, however, is that each elementary rule is now associated to its own specific subset of reactive contexts: these are the only ones the rule can be composed with. These intuitions are formally captured by the following definitions, analogous to those enunciated for RSs.

Definition 3 (Contextual reactive system). *A* contextual reactive system (CRS) *is a* $(\mathcal{C}, I, \mathbf{R}, \mathcal{D}, \mathcal{D}[\![l, r]\!])$ *quintuple, such as* $(\mathcal{C}, I, \mathbf{R}, \mathcal{D})$ *is a* RS*, and* $\mathcal{D}[\![l, r]\!]$ *is a function mapping any elementary rule* $(l, r) \in \mathbf{R}$ *to a composition-reflecting subcategory of* $\mathcal{D}$*.*

Definition 4 (Reaction relationship for contextual reactive systems). *The* reaction relationship*,* $\rightarrow$*, is defined as follows:*

$$a \rightarrow a' \quad \iff \quad \exists\ (l, r) \in \mathbf{R},\ D \in Mo\,(\mathcal{D}[\![l, r]\!])\ .\ a = Dl \wedge a' = Dr$$

As with reactive systems, $\mathbf{R} \subseteq \rightarrow \subseteq \bigcup_{x \in \mathrm{Ob}\,\mathcal{C}} \mathcal{C}(I, x) \times \mathcal{C}(I, x)$.

CRSs differ from RSs only for the presence of a function $\mathcal{D}[\![l, r]\!]$, which captures the association between elementary rules and their allowed reactive contexts, represented by a subcategory of $\mathcal{D}$. Accordingly, the definition of $\rightarrow$ now constrains D to belong to $\mathcal{D}[\![l, r]\!]$, therefore forbidding all the other contexts from being composed with the elementary rule. In CRSs, different elementary rules may have different contextual constraints.

Interestingly, the class of CRSs strictly contains that of RSs. Indeed, the latter are obtained from the former by imposing that $\mathcal{D}[\![l, r]\!] = \mathcal{D}$ for all the $(l, r) \in \mathbf{R}$—i.e., as the CRSs where any elementary rule can be applied under any reactive context. Clearly, CRSs are more expressive, in that the computational locus is now able to influence which interactions can or cannot be performed by the components hosted in it. In a sense, reactive contexts are elevated to a first-class status, and this greatly simplifies the specification task, as discussed in the next section. On the other hand, in our formulation of CRSs an internal transitions performed by a group of components still does *not* affect the surrounding context.

4.2 Elementary Reactive Contexts

The reactive contexts of a RS can usually be expressed as the composition of a number of simpler, non-decomposable contexts. For instance, in the $\mathbf{L}_{\mathrm{RS}}$ system of Table 1, every reactive context is built by repeatedly composing in parallel either terms T, denoting tuples, or terms P, denoting processes. Hence, the set[4] $\mathbf{D}$ of the reactive contexts of $\mathbf{L}_{\mathrm{RS}}$ is generated by all the contexts in the form $- \mid P$ or $- \mid T$. This characteristic is fundamental for enabling a specification of RSs using a rewrite system. Hence, in this section we define formally the notion of elementary context for a RS, and then see how this can be extended to CRSs.

Let us define, for any $\mathbf{X} \subseteq \mathbf{D}$, $\mathbf{X}^\star$ as the minimal set such that $\mathbf{X} \subseteq \mathbf{X}^\star$, and such that $D_0, D_1 \in \mathbf{X}^\star \implies D_0\, D_1 \in \mathbf{X}^\star$, whenever $D_0\, D_1$ is defined. The $\star$ operator is a closure on $\mathbf{D}$, which we call *composition closure*. The following definition is standard in universal algebra:

Definition 5 (Elementary reactive context). *An* (irredundant) basis *for* $\mathbf{D}$ *is a minimal generating set for* $\mathbf{D}$ *w.r.t. the closure operator* $\star$*, i.e., a subset* $\mathbf{B} \subseteq \mathbf{D}$ *such that* $\mathbf{B}^\star = \mathbf{D}$*, and such that* $\mathbf{B}' \subseteq \mathbf{B} \wedge \mathbf{B}'^\star = \mathbf{D} \implies \mathbf{B}' = \mathbf{B}$*. The elements of a basis* $\mathbf{B}$ *for* $\mathbf{D}$ *are called* elementary (reactive) contexts*.*

[4] To improve readability, we use the definitions $\mathbf{D} \stackrel{\mathrm{def}}{=} \mathrm{Mo}\,\mathcal{D}$ and $\mathbf{D}[\![l, r]\!] \stackrel{\mathrm{def}}{=} \mathrm{Mo}\,(\mathcal{D}[\![l, r]\!])$.

When $\mathbf{D}$ has a basis, any reactive context can be finitely decomposed on it, i.e., if $D \in \mathbf{D}$ then $D = B_{n-1} \ldots B_1 B_0$, for some $B_i \in \mathbf{B}$, $i = 0 \ldots n-1$.

The existence of a basis is relevant as it enables us to express the composite rules of a RS by induction. RSs are usually specified by means of a logic, whose axioms are the elementary rules, and whose inference rules describe how composite rules are constructed by composing a (not necessarily elementary) rule with an elementary context. As an example, in $\mathbf{L}_{\mathrm{RS}}$ these inference rules are rule (2.3) and (2.4) for parallel composition in Table 1. On the other hand, composite rules are formally defined as the composition of an elementary rule with a (not necessarily elementary) reactive context. For RSs the two approaches yield the same result, but for CRSs this symmetry is less obvious, because every elementary rule has its own allowed set of reactive contexts, determined by $\mathbf{D}[\![l, r]\!]$. Hence, it makes sense to wonder whether every $\mathbf{D}[\![l, r]\!]$ has a basis. Not surprisingly, the answer is yes, assumed that $\mathbf{D}$ has one. Perhaps a little more surprising is the fact that every $\mathbf{D}[\![l, r]\!]$ has exactly the basis one would expect, namely, the projection of the basis of $\mathbf{D}$ on $\mathbf{D}[\![l, r]\!]$. This is stated formally by the following proposition, which we prove correct.

Proposition 1. *Let* $\mathbf{B}$ *be a basis for* $\mathbf{D}$*, and let us define* $\mathbf{B}[\![l, r]\!] \stackrel{def}{=} \mathbf{D}[\![l, r]\!] \cap \mathbf{B}$*, for each* $(l, r) \in \mathbf{R}$*. Then,* $\mathbf{B}[\![l, r]\!]$ *is a basis for* $\mathbf{D}[\![l, r]\!]$*.*

Proof. We begin by proving that $\mathbf{B}[\![l, r]\!]$ generates $\mathbf{D}[\![l, r]\!]$. $\mathbf{B}$ obviously generates $\mathbf{D}[\![l, r]\!]$, so every context in $\mathbf{D}[\![l, r]\!]$ can be decomposed on $\mathbf{B}$. But $\mathcal{D}[\![l, r]\!]$ is composition-reflecting, thus every component of this decomposition must also be in $\mathbf{D}[\![l, r]\!]$. This proves that every context in $\mathbf{D}[\![l, r]\!]$ can be decomposed on $\mathbf{D}[\![l, r]\!] \cap \mathbf{B} = \mathbf{B}[\![l, r]\!]$, i.e., that $\mathbf{D}[\![l, r]\!] \subseteq (\mathbf{B}[\![l, r]\!])^\star$. Being $\mathbf{B}[\![l, r]\!] \subseteq \mathbf{D}[\![l, r]\!]$, thus $(\mathbf{B}[\![l, r]\!])^\star \subseteq (\mathbf{D}[\![l, r]\!])^\star = \mathbf{D}[\![l, r]\!]$, we proved that $(\mathbf{B}[\![l, r]\!])^\star = \mathbf{D}[\![l, r]\!]$.

Now we prove by contradiction the irredundancy of $\mathbf{B}[\![l, r]\!]$. Let us assume the existence of $\mathbf{B}' \subset \mathbf{B}[\![l, r]\!]$, such as $\mathbf{B}'^\star = \mathbf{D}[\![l, r]\!]$. This means that for some $B \in \mathbf{D}[\![l, r]\!] \cap \mathbf{B}$ is $B \notin \mathbf{B}'$. But $\mathbf{B}'$ generates $\mathbf{D}[\![l, r]\!]$, of which B is a member, thus $B = B_{n-1} \ldots B_1 B_0$, with $B_i \in \mathbf{B}'$. This implies that $\mathbf{B}'' = \mathbf{B} - \{B\} \subset \mathbf{B}$ generates $\mathbf{D}$, thus contradicting the assumption that $\mathbf{B}$ is a basis for $\mathbf{D}$. □

Note that the intersection of a basis for an algebra with the carrier of one of its subalgebras does *not*, in general, yield a basis for the subalgebra. Composition-reflectivity of $\mathbf{D}[\![l, r]\!]$ is the crucial hypothesis yielding this property. This result suggests that expressing composite rules by induction in a CRS is not harder than expressing them in a RS, notwithstanding the proliferation of sets of contexts. This will be the object of the next section.

5 Specifying Contextual Reactive Systems

In this section we move from the abstract, categoric-theoretic setting where we defined CRSs to a process calculi notation suitable for specification. We focus on configurations in the form of *flat* (non-structured) parallel composition of terms. Albeit we omit a formal definition of flat CRS, we remark that all the calculi in this paper are flat, and that flat systems always have a basis for $\mathbf{D}$.

5.1 Basics

A transition systems is usually specified by means of a logic whose sentences express the existence of a (possibly labelled) transition from one state to another. Sentences are built by means of recursive application of inference rules to a set of axioms. Incarnations of this basic principle have been formalized in literature, e.g., by transition systems specifications [10] and rewriting logic [11].

Since RSs differ from CRSs only due to the presence of the $\mathbf{D}[\![l, r]\!]$ function, the important issue to be dealt with in CRSs concerns how composite rules are defined. In a RS specification, appropriate inferences are introduced in the form

$$\frac{a \to a' \quad B \in \mathbf{B}}{B\,a \to B\,a'},$$

i.e., extending rules with elementary contexts. In a CRS, we must attain more strictly to the categorical definition of composite rule and write the inference as

$$\frac{(l, r) \in \mathbf{R} \quad D \in (\mathbf{B}[\![l, r]\!])^\star}{D\,l \to D\,r},$$

because we have no immediate way to recursively extend $\mathbf{D}[\![l, r]\!]$ to composite rules. For this reason, we define a different specification scheme, which evidences how the specification of an elementary rule is associated with the corresponding specification for its $\mathbf{D}[\![l, r]\!]$ set. This scheme has the form:

$$\{B \;.\; \mathcal{P}(B, L, R)\}^\star \qquad L \to R \;.\; \mathcal{P}'(L, R)$$

On the right hand side is the elementary rule scheme $L \to R$, while on the left hand side is a set scheme $\{B \;.\; \mathcal{P}(B, L, R)\}$. The two schemes can be constrained by two predicates $\mathcal{P}$ and $\mathcal{P}'$, respectively. The schemes are instantiated by simultaneous substitution of all their metavariables. This produces an elementary rule $l \to r$ on the right, and a set of elementary contexts on the left. This set is closed with respect to context composition, yielding the set $(\mathbf{B}[\![l, r]\!])^\star = \mathbf{D}[\![l, r]\!]$. Juxtaposition of the set scheme and of the rule scheme stands for their composition, formally defined as $\mathbf{D}[\![l, r]\!]\,(l \to r) \stackrel{\text{def}}{=} \{D\,l \to D\,r \;.\; D \in \mathbf{D}[\![l, r]\!]\}$. This yields the set of all the composite rules generated by the elementary rule $l \to r$. When the elementary rules generated by a given scheme can be applied under any reactive context, we will omit to write the set scheme.

Specifications expressed with this notation are not very readable, since $\mathbf{B}[\![l, r]\!]$ does not bear an intuitive interpretation. Hence, we now present some notational improvements that simplify the specification task and, at the same time, shed a different light on the essence of CRSs.

5.2 Inhibitors and Enablers

Let us reconsider the printer example presented in Section 3. We are now able to define a rule for printing on a raw printer, which does not fire if a PostScript printer is present:

$$\{- \mid \langle v \rangle \;.\; v \neq \text{pr:ps}\}^\star \quad \mathsf{print}(\text{txt}).P \mid \langle \text{pr:raw} \rangle \to P \mid \langle \text{job,txt,raw} \rangle \mid \langle \text{pr:raw} \rangle$$

The set scheme can be rewritten more compactly in terms of its *complement*.

Definition 6 (Inhibiting Elementary Context (Inhibitor)). *Let us define, for any set* $\mathbf{X} \subseteq \mathbf{B}$*, the set* $\mathbf{X}^c \stackrel{def}{=} \mathbf{B} - \mathbf{X}$*. Then,* $\mathbf{I}[\![l, r]\!] \stackrel{def}{=} (\mathbf{B}[\![l, r]\!])^c$ *represents the set of* inhibiting elementary contexts (inhibitors)*, or* anticatalysts*, of the elementary rule* (l, r)*.*

Using inhibitors, we can rewrite the previous rule as:

$$\{- \mid \langle \text{pr:ps} \rangle\}^{c\star} \quad \mathsf{print}(\text{txt}).P \mid \langle \text{pr:raw} \rangle \to P \mid \langle \text{job,txt,raw} \rangle \mid \langle \text{pr:raw} \rangle.$$

This rule yields the same CRS as the previous one, since $(\mathbf{I}[\![l, r]\!])^{c\star} = (\mathbf{B}[\![l, r]\!])^{\star}$. Nevertheless, we argue that the latter is in many cases, including those we illustrated in Section 3, simpler and more intuitive than the former, in that describing what *must not* be present in the configuration at hand, rather than what *may*. This because, in practice, primitives are affected only by a well defined and restricted class of entities, which are precisely those that must not be present, while those that may be present are, indeed, ininfluent.

In the elementary rule scheme for print we considered earlier, we can observe the presence of an invariant part, the tuple $\langle \text{pr:raw} \rangle$. This tuple appears both in the redex and in the contractum of the rule: its presence is necessary for the rule to fire, but the state transition preserves it. In a sense, the tuple represents a portion of the rule context that is nonetheless necessary to its execution. This is a rather common situation, that we can capture formally as follows.

Definition 7 (Maximal Invariant Context). *We informally define the* maximal invariant context (MIC) *of a rule* $l \to r$ *as the reactive context* D *such that* $l = D\,l'$ *and* $r = D\,r'$ *for some pair of ground contexts* l', r'*, and such that the new rule* $l' \to r'$ *is* minimal*, i.e. does not preserve in its contractum any of the entities appearing on its redex.*

Minimality can be formalized as the requirement that l' and r' are *relatively prime*, i.e., their decompositions on $\mathbf{B}$ have no elementary context in common besides the $-$. The MIC of the rule for print is $- \mid \langle \text{pr:ps} \rangle$. It is composed of a single elementary context, although in general the MIC of a rule may contain more than one. For uniformity, rather than exploiting directly the MIC in the specification we prefer to deal with its elementary contexts:

Definition 8 (Enabling Elementary Context (Enabler)). *The* enabling elementary contexts (enabler)*, or* catalysts*, of a rule are all the elementary contexts belonging to the rule's* MIC*.*

When enablers and inhibitors are both put in evidence, a specification becomes:

$$\{I \in \mathbf{B} \;.\; \mathcal{P}(I, L, R)\}^{c\star} \quad \{\!|E \in \mathbf{B} \;.\; \mathcal{P}'(I, L, R)|\!\}^{;} \quad L \to R \;.\; \mathcal{P}''(L, R)$$

with the constraint that the rules defined by the scheme $L \to R \;.\; \mathcal{P}''(L, R)$ must be minimal. The multiset scheme $\{\!|E \in \mathbf{B} \;.\; \mathcal{P}'(I, L, R)|\!\}$ specifies the enablers, and the semicolon operator (;) applied on a multiset of contexts, yields the composition of all the contexts in the multiset[5]. Using this notation, the

[5] This definition is unambiguous if $E_0\,E_1 = E_1\,E_0$ when both exist. For the systems we are considering this is precisely commutativity of parallel composition.

example rule for the print operation becomes:

$$\{- \mid \langle \text{pr:ps} \rangle\}^{c\star}\ \{\!|\ - \mid \langle \text{pr:raw} \rangle |\!\}^{;}\ \mathsf{print}(\text{txt}).P \to P \mid \langle \text{job,txt,raw} \rangle,$$

where the semicolon exponent is, in this case, pleonastic.

Enablers and inhibitors are similar: both determine how a behavior is affected by the surrounding environment. Nevertheless, they have different effects on the rule they are associated to. Inhibitors specify the set of elementary contexts that *must not* be present in a rule's context: their presence disables the firing of the rule. On the other hand, enablers define the set of elementary contexts that *must* be present in the context of a rule for it to fire. The next section discusses how these abstractions can be used effectively to model a real middleware.

6 An Example: Formalizing LIME

In this section we set out to specify the $\mathbf{Li}_{\text{CS}}$ system, a subset of the LIME coordination middleware [3]. We first summarize informally its main features, and then show how it can be given a formal semantics effectively by using a CRS.

In $\mathbf{Li}_{\text{CS}}$ processes and tuples are organized in *agents*, the units of mobility. Agents may be members of *groups*. Group membership defines the set of tuples accessible to an agent. The tuples of an agent are transiently shared throughout the group the agent is member of: an in on the tuple space may return a local tuple as well as one belonging to another agent in the group. An agent joins a group through an *engagement* procedure, and leaves it through a *disengagement*. As the agent disengages, all its tuples become unavailable to the group. Insertion of a tuple in the tuple space is performed through a modified version of out, annotated with the name of a destination agent. If source and destination are engaged in the same group, the tuple is immediately delivered. Otherwise, it is retained at the source and tagged with the name of its destination; these tuples are said to be *misplaced*. Misplaced tuples are delivered as soon as the source and the destination agents become engaged in the same group.

The syntax of terms is shown at the top of Table 2. The A term denotes the existence of an agent in a configuration, and has the form $g :: a$, where a is the name of the agent and g is the name of the group it belongs to. The special name ϵ is used to indicate disengaged agents, i.e., agents that are not members of any group. Tuples and processes are qualified with the agent they belong to. Tuples also report, after the @ symbol, the name of their destination agent[6]. The out operation is annotated with the name of the destination agent. The engagement en and disengagement den operations, together with in, complete the calculus.

The semantics of the calculus is reported in the rest of Table 2. An evident difference with conventional calculi is that the specification is split in two parts: behavioral rules and the (enabling and inhibiting) contexts constraining their execution. Structural congruence on terms is defined as for $\mathbf{L}_{\text{RS}}$, with the equation $C \mid 0 \equiv C$ replaced by $C \mid a :: 0 \equiv C \iff C \equiv C' \mid g :: a$.

[6] To improve readability, this is left out when irrelevant.

Syntax:

$$C ::= A \mid P \mid T \mid C \,|\, C \qquad A ::= \epsilon :: a \mid g :: a$$
$$P ::= a :: 0 \mid a :: X \mid a :: c.P \mid a :: \mathsf{rec}X.P \qquad T ::= a :: \langle v \rangle @\, a$$
$$c ::= \mathsf{in}(v) \mid \mathsf{out}[a](v) \mid \mathsf{en}(g) \mid \mathsf{den}$$

Rules:

$$a :: \mathsf{in}(v).P \,|\, a :: \langle v \rangle \to P \tag{6.1}$$
$$\mathbf{E}_1^{\mathsf{i}} \quad a :: \mathsf{in}(v).P \,|\, a' :: \langle v \rangle \to P \tag{6.2}$$
$$\mathbf{E}_2^{\mathsf{i}} \quad a :: \mathsf{out}[a'](v).P \to a :: P \,|\, a :: \langle v \rangle @\, a' \tag{6.3}$$
$$\mathbf{I}_1^{\mathsf{c}\star}\ \mathbf{E}_3^{\mathsf{i}} \quad a :: \mathsf{out}[a'](v).P \to a :: P \,|\, a :: \langle v \rangle @\, a' \tag{6.4}$$
$$\mathbf{E}_1^{\mathsf{i}} \quad a :: \mathsf{out}[a'](v).P \to a :: P \,|\, a' :: \langle v \rangle @\, a' \tag{6.5}$$
$$a :: \mathsf{den}.P \,|\, g :: a \to a :: P \,|\, \epsilon :: a \tag{6.6}$$
$$\mathbf{I}_2^{\mathsf{c}\star}\ \mathbf{E}_4^{\mathsf{i}} \quad a :: \mathsf{en}(g).P \,|\, \epsilon :: a \,|\, \prod_{\mathbf{T}_a} a_i :: \langle v_i \rangle @\, a \to a :: P \,|\, g :: a \,|\, \prod_{\mathbf{T}_a} a :: \langle v_i \rangle @\, a$$
$$. \; a' :: \langle v_i \rangle @\, a \in \mathbf{T}_a \implies g :: a' \in \mathbf{A}_g \tag{6.7}$$
$$\frac{D \equiv C \quad C \to C' \quad C' \equiv D'}{D \to D'} \tag{6.8}$$

Contexts:

$$\mathbf{E}_1 = \{\!|\, - \,|\, g :: a,\, - \,|\, g :: a' \;.\; a \neq a' \,|\!\} \qquad \mathbf{E}_2 = \{\!|\, - \,|\, \epsilon :: a \,|\!\}$$
$$\mathbf{E}_3 = \{\!|\, - \,|\, g :: a \,|\!\} \qquad \mathbf{E}_4 = \{\!|\, - \,|\, g :: a' \;.\; g :: a' \in \mathbf{A}_g \,|\!\}$$
$$\mathbf{I}_1 = \{- \,|\, g :: a'\} \qquad \mathbf{I}_2 = \begin{array}{l}\{- \,|\, g :: a' \;.\; g :: a' \notin \mathbf{A}_g \;\} \cup \\ \{- \,|\, a' :: \langle v \rangle @\, a \;.\; g :: a' \in \mathbf{A}_g\}\end{array}$$

Table 2. Syntax and semantics of the $\mathbf{Li}_{\mathrm{CS}}$ system.

The first two rules define the semantics of in. Rule (6.1) defines input of local tuples, i.e., belonging to the same agent a that issued the operation. Rule (6.2) specifies an in withdrawing the tuple from the tuple space of a different agent a', by virtue of transient sharing. In this case, $\mathbf{Li}_{\mathrm{CS}}$ prescribes that the two agents a and a' must be members of the same group. This requirement is captured[7] by the enabling context $\mathbf{E}_1$. The out primitive is described by three rules. Rule (6.3) defines the semantics of an out issued by a disengaged agent: the emitted tuple is retained locally. Rule (6.4) specifies an out issued by an engaged agent for a destination agent currently not engaged in the same group. Notably, the two rules are exactly the same: the different semantics is entirely captured by the contexts associated to the rules. In the first case, the enabler $\mathbf{E}_2$ requires that

[7] The two rules could be collapsed in a one by removing the predicate in $\mathbf{E}_1$. We chose to model them separately to highlight the fact that one is local and the other remote.

the agent is disengaged. In the second case, an enabler and an inhibitor are combined to express that the agent must be engaged in g, and the destination agent a' must *not* be engaged in the same group. Finally, rule (6.5) specifies the immediate delivery of the tuple when the source and destination agent are both engaged in g. Again, this rule is very similar to the other two, and actually leverages the enabler $\mathbf{E}_1$ previously used by rule (6.2).

This first fragment of the semantics of $\mathbf{Li}_{\mathrm{CS}}$ evidences the significant advantages brought by our approach. The specification of behavior is sharply decoupled from the specification of the context where it may occur. This greatly simplifies the understanding of the specification, because it allows to grasp more directly the similarity between rules. In many cases it even elicits the fact that two rules with different semantics are in reality the same behavior applied in two different contexts (as with out), or two different behaviors taking place in the same context (as with $\mathbf{E}_1$). A specification with a traditional calculus is bound to bury these fundamental aspects in a direct representation of context, and hence obfuscates irremediably the true meaning of the semantic rules.

The specification of $\mathbf{Li}_{\mathrm{CS}}$ is completed by the semantics of engagement and disengagement. Disengagement is performed simply by changing the agent's group name in rule (6.6). Engagement must also perform delivery of misplaced tuples to the agent a joining the group. Rule (6.7) relies on the definition of the sets $\mathbf{T}_a$, containing all the misplaced tuples in the system that are destined to a, and $\mathbf{A}_g$, containing the names of all the agents engaged in g. The redex describes a situation where a is currently disconnected and the system contains several misplaced tuples destined for it. The contractum shows a engaged in g and containing all misplaced tuples. The predicate specifies that the misplaced tuples in $\mathbf{T}_a$ must belong to some agent in g. The enabler $\mathbf{E}_4$ ensures that all the terms representing the agents currently engaged are present, with the twofold purpose of ensuring that the tuples in $\mathbf{T}_g$ effectively belong to some agent in g, and that all of them are involved in the engagement procedure. The inhibitor $\mathbf{I}_2$ forces two global constraints: $\mathbf{A}_g$ must contain *all* the agents in g, and $\mathbf{T}_a$ must contain *all* the misplaced tuples in the g with the engaging agent a as destination. This prevents the rule from firing in contexts where some agents and/or tuples are "left behind" during engagement.

The specification of engagement is arguably more complex and less intuitive than the one for the other operations. However, some observations are noteworthy. First, engagement is the most complex portion of the original LIME model, both in terms of semantics and actual middleware implementation. Then, it is not surprising to see such complexity reflected in the specification. Second, dealing with misplaced tuples and their reconciliation essentially involves dealing explicitly with state—something process calculi are notoriously not very good at. Third, without the level of abstraction provided by explicitly separating context, the specification would have been *much* more complicated and awkward. This is best understood by observing that the problem solved by the inhibitor $\mathbf{I}_2$ (i.e., ensuring that the rule fires in a context with *all* the agents and *all* their misplaced tuples) is very similar to the problem of implementing ing we

discussed in Section 3. In our approach, the problem of ensuring that *all* of the necessary context is present and of preventing firing in all of the "partial" context configurations finds a natural and elegant solution in the use of inhibitors.

7 Discussion

To our knowledge, none of the works in the lively field of calculi for distributed systems takes a perspective similar to ours. From a strictly technical point of view, the closest approach is described in [9], which gives a semantics for a Linda-like calculus with reactive programming constructs. However, all the rules are specified directly, with the allowed contexts explicitly mentioned in the rule schemes. This solution yields a RS somewhat similar to a CRS, although the authors do not strive towards a general framework, and hence the resulting model suffers from the drawbacks we discussed at the beginning of Section 4.1.

Since our study is at an early stage, the opportunities for further research are many, and we cite only some here. Firstly, the formal properties of CRSs are largely to be assessed. We are currently investigating if desirable operational congruences can be obtained by extending the technique in [6], which builds labelled transition systems with bisimulations as congruences from some classes of RSs. Extending them to CRSs to obtain operational congruence is apparently harder, since congruence in a CRS may be broken not only because new behaviors may arise by augmenting the context, but also because old ones may be disabled. We conjecture that the extension of this technique is likely to resemble transition systems with negative premises [12]. Secondly, we want to address arbitrary models of space, beyond flat (as in $\mathbf{Li}_{\mathrm{CS}}$) or hierarchical (as in the Ambient calculus) ones. Examples are spaces with adjacency, or multiple coexisting descriptions, e.g., combining physical and logical notions of space. Reflecting the space structure in the syntactic structure of the calculus terms, as done in Ambients, is no longer feasible. We are therefore studying how to represent spatial information in a flat system. Finally, our long-term goal is to define an effective and usable specification framework. In this context, a CRS logic is a viable option to increase expressiveness and improve decoupling of the operational and the context-related parts of the specification. Formalization of other coordination models and middleware will be a necessary step to validate our approach.

8 Conclusions

In modern distributed computing the modeling of a dynamically changing context is becoming of paramount importance. Unfortunately, process calculi, a common formal tool for describing concurrent and distributed systems, do not provide specialized abstraction to deal effectively with context. In this paper, we set the grounds for *contextual reactive systems* (CRS), a generalization of the well-known reactive systems where the representation of context is sharply decoupled from the one of processes. We maintain that the abstraction characteristics of our framework facilitate the specification chore and improve the understanding

of the resulting specification. Evidence for our hypothesis is provided in this paper by defining suitable process calculi abstractions based on our categorical formulation of CRSs, and by presenting as an example the formalization of a subset of LIME, a coordination middleware for mobile computing.

Acknowledgements. The work described in this paper was supported by the project NAPI, funded by Microsoft Research Cambridge, and partially supported by the projects SAHARA, VICOM and IS-MANET, funded by the Italian government. The authors would like to thank Alessandra Cherubini for her comments on early drafts of this paper.

References

1. Roman, G.C., Murphy, A., Picco, G.: Coordination and Mobility. In Omicini, A., Zambonelli, F., Klusch, M., Tolksdorf, R., eds.: Coordination of Internet Agents: Models, Technologies, and Applications. Springer (2000) 254–273
2. Cardelli, L., Gordon, A.: Mobile ambients. In Nivat, M., ed.: Proc. of the 1^{st} Int. Conf. on Foundations of Software Science and Computation Structures (FoSSaCS'98). Volume LNCS 1378., Springer (1998) 140–155
3. Murphy, A., Picco, G., Roman, G.C.: LIME: A Middleware for Physical and Logical Mobility. In Golshani, F., Dasgupta, P., Zhao, W., eds.: Proc. of the 21^{st} Int. Conf. on Distributed Computing Systems (ICDCS-21). (2001) 524–533
4. Murphy, A., Picco, G., Roman, G.C.: LIME: A Coordination Middleware Supporting Mobility of Hosts and Agents. Technical report, Politecnico di Milano (2003) Submitted for publication. Available at `www.elet.polimi.it/~picco`.
5. Berry, G., Boudol, G.: The Chemical Abstract Machine. Theoretical Computer Science **96** (1992) 217–248
6. Leifer, J., Milner, R.: Deriving bisimulation congruences for reactive systems. In Palamidessi, C., ed.: Proc. of the Int. Conf. on Concurrency Theory (CONCUR 2000). LNCS 1187, Springer (2000) 243–258
7. Sassone, V., Sobociński, P.: Deriving Bisimulation Congruences: A 2-Categorical Approach. In Nestmann, U., Panangaden, P., eds.: Proc. of the 9^{th} Int. Wksp. on Expressiveness in Concurrency (EXPRESS'02). Volume 68 of Electronic Notes in Theoretical Computer Science., Springer (2002)
8. Plotkin, G.: Call-by-name, call-by-value and the λ-calculus. Theoretical Computer Science **1** (1975) 125–159
9. Busi, N., Rowstron, A., Zavattaro, G.: State- and event-based reactive programming in shared dataspaces. In Arbab, F., Talcott, C., eds.: Proc. of the 5^{th} Int. Conf. on Coordination Models and Languages (COORDINATION'02). LNCS 2315, Springer (2002) 111–124
10. Groote, J., Vaandrager, F.: Structured operational semantics and bisimulation as a congruence. Information and Computation **100** (1992) 202–260
11. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theoretical Computer Science **96** (1992) 73–155
12. Groote, J.: Transition system specifications with negative premises. Theoretical Computer Science **118** (1993) 263–299