Synchronised Hyperedge Replacement as a Model for Service Oriented Computing*

Gian Luigi Ferrari[1], Dan Hirsch[2], Ivan Lanese[3], Ugo Montanari[1], and Emilio Tuosto[4]

[1] Computer Science Department, University of Pisa, Italy
{giangi, ugo}@di.unipi.it
[2] Computer Science Department, University of Pisa, Italy and Department of Computing, Imperial College, London, UK
dhirsch@doc.ic.ac.uk
[3] Computer Science Department, University of Bologna, Italy
lanese@cs.unibo.it
[4] Computer Science Department, University of Leicester, UK
et52@mcs.le.ac.uk

Abstract. This tutorial paper describes a framework for modelling several aspects of distributed computing based on *Synchronised Hyperedge Replacement* (SHR), a graph rewriting formalism. Components are represented as edges and they rewrite themselves by synchronising with neighbour components the *productions* that specify their behaviour. The SHR framework has been equipped with many formal devices for representing complex synchronisation mechanisms which can tackle mobility, heterogeneous synchronisations and non-functional aspects, key factors of Service Oriented Computing (SOC). We revise the SHR family as a suitable model for contributing to the formalisation of SOC systems.

1 Introduction

Modern distributed inter-networking systems are very complex and constituted by a varied flora of architectures and communicating infrastructures. Such systems are heterogeneous, geographically distributed and highly dynamic since the communication topology can vary and the components can, at any moment, connect to or detach from the system. Recently, *Service Oriented Computing* (SOC) has emerged as a suitable paradigm for specifying such global systems and applications. Engineering issues are tackled by exploiting the concept of *services*, which are the building blocks of systems. Services are autonomous, platform-independent, mobile/stationary computational entities. In the deployment phase, services can be independently described, published and categorised. At runtime they are searched/discovered and dynamically assembled for building wide area distributed systems.

All this requires, on the one hand, the development of foundational theories to cope with the requirements imposed by the global computing context, and, on the other hand, the application of these theories for their integration in a pragmatic software engineering approach. At the architectural level, the fundamental features to take into account for the description of components and their interactions include: dynamic (possibly self

* Partially supported by the Project EC FET – Global Computing 2, IST-2005-16004 SENSORIA.

F.S. de Boer et al. (Eds.): FMCO 2005, LNCS 4111, pp. 22–43, 2006.
© Springer-Verlag Berlin Heidelberg 2006

organising) reconfiguration, mobility, coordination, complex synchronisation mechanisms, and awareness of *Quality of Service* (QoS).

Process calculi are among the most successful models for concurrency and, in the last years, CSP [19], CCS [26] and π-calculus [27] gained paramount relevance and helped to understand many of the phenomena arising in distributed computing. Many of the recent proposals like Ambient [2], Klaim [8], Join [12] and D-π [30] (to cite a few) have been deeply inspired by the work on CSP, CCS and π-calculus. Also, distributed systems can be naturally modelled by means of graph-based techniques [32]. Among those, we choose *Synchronised Hyperedge Replacement* (SHR) where systems are modelled as *hypergraphs*, that is graphs where each (hyper)edge can be connected to any number of nodes (instead of just two). Edges represent components connected via shared nodes (representing communication ports).

Originally, SHR aimed at modelling distributed systems and software architectures, however, it turns out to be expressive enough to model many process calculi. In fact, it can naturally encode π-calculus [14], Ambient and Klaim [33] or Fusion [23]. In our opinion, SHR conjugates the ability of expressing various forms of synchronisation and communication features (typical of process calculi) with a suggestive visual representation of systems' topology (typical of graph models). In SHR, constraint satisfaction is exploited to guide rewriting by synchronising context-free *productions* that specify the behaviour of single edges. Productions define how an edge can be rewritten into a generic graph and the conditions that this rewriting imposes on adjacent nodes. Global transitions are obtained by parallel application of productions with "*compatible*" conditions. What "compatible" exactly means depends on the chosen synchronisation model. The Hoare model (so called since it extends CSP synchronisation [19]), for instance, requires that all edges connected to the same node execute the same action on it. Instead, the Milner model (extending the model of CCS [26]) requires exactly two edges to interact by performing complementary actions while the other edges must stay idle on that node. SHR, and in particular its variant SHR-HS [25] (outlined in § 7), allows also different synchronisation policies to live together in a single framework.

Aims and structure of the paper. A number of published results (see the brief bibliographic note at the end of this section) is here collected with the main goal to give a systematic presentation of the SHR approach. A relevant effort has indeed been put on giving a uniform and incremental presentation. Also, we tried to help intuition by showing how the various synchronisation mechanisms actually extend the basic model discussed in § 3. It might be useful to have the many versions of SHR harmonised within a common formal context and we hope to have been able to clearly introduce the SHR family by rephrasing it in simpler, yet rigorous, definitions.

Preliminary definitions and notations for graphs are reported in § 2. We introduce *basic Milner SHR* (bMSHR for short) in § 3, where the mathematical basis of SHR are discussed in the simpler framework based on Milner synchronisation without considering name mobility and name fusion. These aspects are added in § 4, giving rise to MSHR. This extension allows to substantially increase the expressivity of the approach for modelling both architectural and programming aspects of mobile and reconfigurable distributed applications. In § 5 we define *Synchronisation Algebras with Mobility* (SAMs for short), an abstract formalisation of the concept of synchronisation

model, extending Winskel's synchronisation algebras (SAs) [35] to cope with mobility and handling of local resources. SAMs are exploited in § 6, where we present *parametric SHR* [24,22] which permits to abstract from the synchronisation model by choosing each time the most adequate SAM (whose primitives correspond to the ones used in the modelled system). Parametric SHR smoothly adapts SHR to various interaction mechanisms; for instance, it can uniformly represent MSHR and SHR with Hoare synchronisation. A first SAM-based SHR is *SHR for heterogeneous systems* (SHR-HS) [25,21] in § 7 where different SAMs can be associated to different nodes. SHR-HS has been devised to model systems where heterogeneity concerns both applications and their underlying middlewares so that different synchronisation policies can be used and dynamically changed (and, hence, negotiated) within systems. This feature is fundamental to model coordination at the application level, where interaction patterns are dynamically determined. Another SAM-based SHR proposal is SHReQ [17] (§ 8), an SHR framework for handling abstract high-level QoS requirements expressed as *constraint-semirings* (c-semirings) [1], algebraic structures suitable for multi-criteria QoS [6]. We exploit the algebraic features of c-semirings by embedding them in the SHR synchronisation mechanism: interactions among components are ruled by synchronising them on actions that are c-semiring values, expressing QoS constraints imposed by all components participating to the synchronisation. Finally, in § 9 we outline our plans for future investigations.

Brief SHR bibliography. Various facets of SHR have been studied w.r.t. issues related to distributed systems. SHR has been introduced in [3] with the name of "Grammars for Distributed Systems". Here Hoare synchronisation was used, and the emphasis was on analysing the history of the computation, explicitly represented as part of the graph. Infinite computations and concurrency issues have been considered in [9] while [4] extends SHR by allowing to merge and split nodes. In [31] there is a presentation of SHR inside the Tile Model [13], and an approach to find the allowed transitions using constraint solving techniques is also proposed. A main extension is given in [15], where *node mobility* is added. This is obtained by allowing actions to carry tuples of nodes. When actions synchronise the carried tuples of nodes are merged. This allows to create new connections at runtime. In literature, inference rules (in the SOS style [29]) based on a notation for representing graphs as *syntactic judgements* are defined for different mobile synchronisation mechanisms, presenting SHR as a general model for mobile process calculi. However, SHR extends process algebras to allow synchronisations of any number of partners and on any number of channels at the same time. In [15] only newly created nodes can be communicated and merged. In [20] a mapping into the Tile Model [13] is used to prove that an ad hoc bisimilarity is a congruence. Another important step is made in [16], where also old nodes can be communicated, but they can be merged only with new nodes. This kind of SHR, with Milner synchronisation, is shown to be strictly related [16] to π-calculus [27]. A later improvement is presented in [10], where fusions of arbitrary nodes are allowed. These are exploited [10,23] to give semantics to the Ambient Calculus [2] and to the Fusion Calculus [28]. Finally, in [24,22] the SHR synchronisation mechanism is generalised allowing a complete parametrisation of SHR w.r.t. the synchronisation and mobility policies. Many applications of SHR can be found in literature, in particular in the field of process calculi [33], of software architectures [14,15,5] and QoS [17].

2 Hypergraphs

In this section we introduce a presentation of (hyper)graphs as (syntactic) judgments, which is convenient to write the rules for describing SHR behaviour. We first introduce some mathematical notations.

Notation. Given a set V, we let V^* be the set of tuples on V. We denote a tuple as $\mathbf{v} = \langle v_1, \ldots, v_n \rangle$, the empty tuple as $\langle\rangle$, the i-th element of $\mathbf{v}$ as $\mathbf{v}[i]$, and write $|\mathbf{v}|$ for the length of $\mathbf{v}$.

Given a function f, $\mathrm{dom}(f)$ is its domain, and function $f\lfloor_S$ is the restriction of f to S, namely $f\lfloor_S(x) = f(x)$ if $x \in S$, $f\lfloor_S(x)$ is undefined otherwise. We denote with $f \circ g$ the composition of f and g, namely $(f \circ g)(x) = f(g(x))$.

For a syntactic structure s with names and binders, $\mathrm{fn}(s)$ is the set of its free names.

A graph is composed by a set of nodes and a set of *(hyper)edges* which connect nodes. Set $\mathcal{N}$ is a countable infinite set of node names while set $\mathcal{L}$ is the set of edge labels. A label $L \in \mathcal{L}$ is assigned a *rank*, i.e., a natural number (denoted as $\mathrm{rank}(L)$). An edge labelled by L connects $\mathrm{rank}(L)$ nodes and a node connected to an edge is said to be an *attachment node* of that edge.

A *syntactic judgment* specifies a graph along with its interface, i.e., its *free nodes*.

Definition 2.1 (Graphs as judgements). *A* judgment *has form* $\Gamma \vdash G$ *where:*

1. $\Gamma \subseteq \mathcal{N}$ *is a finite set of names (the free nodes of the graph);*
2. G *is a* graph term *generated by the grammar*

$$G ::= L(\mathbf{x}) \quad | \quad G|G \quad | \quad \nu y\, G \quad | \quad nil$$

where $\mathbf{x}$ *is a tuple of names,* $L \in \mathcal{L}$, $\mathrm{rank}(L) = |\mathbf{x}|$ *and* y *is a name.*

In $\nu y\, G$*, restriction operator* ν *binds* y *in* G*,* $\mathrm{fn}(G)$ *is defined accordingly as usual and we demand that* $\mathrm{fn}(G) \subseteq \Gamma$.

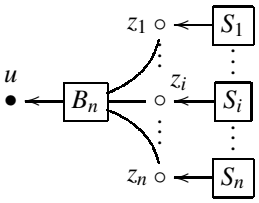
Graph *nil* is the empty graph, $|$ is the parallel composition operator of graphs (merging nodes with the same name) and νy is the restriction operator of nodes; free/bound nodes correspond to free/bound names. Edges are terms of the form $L(x_1, \ldots, x_n)$, where the x_i are arbitrary names and $\mathrm{rank}(L) = n$. Condition $\mathrm{fn}(G) \subseteq \Gamma$ accounts for having free isolated nodes in G (e.g., $\{x\} \vdash nil$ is graph with only the isolated node x).

We assume that restriction has lower priority than parallel composition. For conciseness, curly brackets are dropped from interfaces Γ in judgements and Γ_1, Γ_2 denotes $\Gamma_1 \cup \Gamma_2$, provided that $\Gamma_1 \cap \Gamma_2 = \emptyset$ (e.g., $\Gamma, x = \Gamma \cup \{x\}$, if $x \notin \Gamma$).

Example 2.2. Consider the judgment

$$u \vdash \nu z_1, \ldots, z_n\; B_n(u, z_1, \ldots, z_n) | S_1(z_1) | \ldots | S_n(z_n)$$

which describes a system where many servers S_i are connected to the network via a manager B_n and can be graphically represented as:



Edges are drawn as rectangles and nodes are bullets (empty for bound nodes and solid for free nodes). A connection between a node and an edge is represented by a line, called *tentacle*; an arrowed tentacle indicates the first attachment node of the edge. The other nodes are determined by numbering tentacles clockwise (e.g., for B_n, u is the first attachment node, z_1 is the second and so on).

Definition 2.3 (Structural congruence on graph judgements). *Graph terms are considered up to axioms* $(AG1 \div 7)$ *below:*

$$(\texttt{AG1})\ (G_1|G_2)|G_3 \equiv G_1|(G_2|G_3) \quad (\texttt{AG2})\ G_1|G_2 \equiv G_2|G_1 \quad (\texttt{AG3})\ G|nil \equiv G$$
$$(\texttt{AG4})\ \nu x\, \nu y\, G \equiv \nu y\, \nu x\, G \quad (\texttt{AG5})\ \nu x\, G \equiv G \textit{ if } x \notin \mathrm{fn}(G)$$
$$(\texttt{AG6})\ \nu x\, G \equiv \nu y\, G\{y/x\},\ \textit{if } y \notin \mathrm{fn}(G) \quad (\texttt{AG7})\ \nu x\, G_1|G_2 \equiv G_1|\nu x\, G_2,\ \textit{if } x \notin \mathrm{fn}(G_1)$$

For judgments, we define $\Gamma_1 \vdash G_1 \equiv \Gamma_2 \vdash G_2$ *iff* $\Gamma_1 = \Gamma_2$ *and* $G_1 \equiv G_2$.

Axioms (AG1), (AG2) and (AG3) define respectively the associativity, commutativity and identity over *nil* for operator $|$. Axioms (AG4) and (AG5) state that nodes can be restricted only once and in any order. Axiom (AG6) defines α-conversion of a graph w.r.t its bound names. Axiom (AG7) defines the interaction between restriction and parallel composition (note that function fn is well-defined on equivalence classes). We consider judgements for graphs up to structural congruence which amounts to consider graphs up to graph isomorphisms that preserve free nodes, labels of edges, and tentacles [14].

3 Basic Milner SHR

The simplest version of SHR is *basic Milner SHR* (bMSHR), where "basic" refers to the absence of mobility and "Milner" is reminiscent of the CCS synchronisation. Later, bMSHR will be extended with mobility and more complex synchronisation policies.

Milner synchronisation models two-parties synchronisation and requires that actions are partitioned into normal actions a and co-actions $\overline{a}$ (where $\overline{\overline{a}} = a$). Furthermore, there are two special actions: an action ε standing for "not taking part to the synchronisation" and an action τ representing a complete binary synchronisation. Thus, Milner synchronisation on a node x requires two complementary actions to interact, while other connected edges must stay idle on x (i.e., they all exhibit action ε on x). The final result of the synchronisation is τ.

Notation. A *renaming* is a function $\sigma : \mathcal{N} \to \mathcal{N}$, $x\sigma$ is the application of σ to $x \in \mathrm{dom}(\sigma)$ and yields $\sigma(x)$. If $\sigma \circ \sigma = \sigma$, the renaming is said *idempotent* and is *injective* when σ is injective. Renaming $\{x/y\}$ is such that $\{x/y\}(y) = x$ and $\{x/y\}(z) = z$ for all $z \neq y$ in the domain of $\{x/y\}$.

We use $\{(x,y) \mid x \in \mathrm{dom}(f) \ \wedge\ y = f(x)\}$ as a set-theoretic representation of a function f.

We can now define *transitions*, and then we show some inference rules to derive them from *productions*.

Definition 3.1 (SHR transitions). *A relation* $\Gamma \vdash G \xrightarrow{\Lambda} \Gamma \vdash G'$ *is an* SHR transition *if* $\Gamma \vdash G$ *and* $\Gamma \vdash G'$ *are judgments for graphs, and* $\Lambda : \Gamma \to Act$ *is a total function, where* Act *is a set of actions.*

Intuitively transition $\Gamma \vdash G \xrightarrow{\Lambda} \Gamma \vdash G'$ specifies that graph $\Gamma \vdash G$ is rewritten into $\Gamma \vdash G'$ and, while doing this, the action $\Lambda(x)$ is performed on each node x in the interface Γ. Notice that the starting and the final graph share the same interface.

Productions are special transitions specifying the behaviour of a single edge.

Definition 3.2 (Productions). *A* production *is an SHR transition of the form:*

$$x_1,\ldots,x_n \vdash L(x_1,\ldots,x_n) \xrightarrow{\Lambda} x_1,\ldots,x_n \vdash G \tag{1}$$

where $\mathrm{rank}(L) = n$ *and* $x_1,\ldots,x_n$ *are all distinct. Production (1) is* idle *iff* $\Lambda(x_i) = \varepsilon$ *for each* i *and* G *is* $L(x_1,\ldots,x_n)$.

A transition is obtained by composing productions in a set $\mathcal{P}$ that contains any idle production and is closed under all injective renamings (that is, the application of an injective renaming to a productions in $\mathcal{P}$ yields productions in $\mathcal{P}$).

Composition is performed by merging nodes and thus connecting the edges. Synchronisation conditions as specified in productions must be satisfied.

Definition 3.3 (Inference rules for bMSHR). *The admissible behaviours of bMSHR are defined by the following inference rules.*

$$\textit{(par-b)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda} \Gamma \vdash G_2 \qquad \Gamma' \vdash G_1' \xrightarrow{\Lambda'} \Gamma' \vdash G_2' \qquad \Gamma \cap \Gamma' = \emptyset}{\Gamma,\Gamma' \vdash G_1|G_1' \xrightarrow{\Lambda\cup\Lambda'} \Gamma,\Gamma' \vdash G_2|G_2'}$$

$$\textit{(merge-b)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda} \Gamma \vdash G_2}{\Gamma\sigma \vdash G_1\sigma \xrightarrow{\Lambda'} \Gamma\sigma \vdash G_2\sigma}$$

where $\sigma : \Gamma \to \Gamma$ *is an idempotent renaming and:*

1. *for all* $x,y \in \Gamma$ *such that* $x \neq y$*, if* $x\sigma = y\sigma$*,* $\Lambda(x) \neq \varepsilon$ *and* $\Lambda(y) \neq \varepsilon$ *then* $(\forall z \in \Gamma \setminus \{x,y\}.z\sigma = x\sigma \Rightarrow \Lambda(z) = \varepsilon) \wedge \Lambda(x) = a \wedge \Lambda(y) = \overline{a} \wedge a \neq \tau$
2. $\Lambda'(z) = \begin{cases} \tau & \textit{if } x\sigma = y\sigma = z \wedge x \neq y \wedge \Lambda(x) \neq \varepsilon \wedge \Lambda(y) \neq \varepsilon \\ \Lambda(x) & \textit{if } x\sigma = z \wedge \Lambda(x) \neq \varepsilon \\ \varepsilon & \textit{otherwise} \end{cases}$

$$\textit{(res-b)} \quad \frac{\Gamma,x \vdash G_1 \xrightarrow{\Lambda} \Gamma,x \vdash G_2 \qquad \Lambda(x) = \varepsilon \vee \Lambda(x) = \tau}{\Gamma \vdash \nu x\, G_1 \xrightarrow{\Lambda|_\Gamma} \Gamma \vdash \nu x\, G_2}$$

$$\textit{(new-b)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda} \Gamma \vdash G_2 \qquad x \notin \Gamma}{\Gamma,x \vdash G_1 \xrightarrow{\Lambda\cup\{(x,\varepsilon)\}} \Gamma,x \vdash G_2}$$

Rule (par-b) deals with the composition of transitions which have disjoint sets of nodes and rule (merge-b) allows to merge nodes. Condition 1 requires that at most two non ε actions are performed on nodes to be merged. If they are exactly two then they have to be complementary, and the resulting action is τ (condition 2). Since σ is required to be idempotent, it yields an equivalence relation on Γ and a choice of a standard representative. In fact, $x,y \in \Gamma$ are equivalent under σ iff $x\sigma = y\sigma$; the representative element

of the equivalence class of x is $x\sigma$. Rule (res-b) binds node x. This is allowed only if either τ or ε actions are performed on x, forcing either a complete synchronisation (τ) or no synchronisation (ε). Rule (new-b) allows to add to the source graph an isolated free node where an action ε is performed.

Example 3.4. Consider an instance of the system in Example 2.2 where edge $B_2(u,z_1,z_2)$ takes requests on node u and broadcasts them to $S_1(z_1)$ and $S_2(z_2)$ by synchronising on nodes z_1 and z_2, respectively. The productions for B_2 and S_i ($i \in \{1,2\}$) are:

$$u,z_1',z_2' \vdash B_2(u,z_1',z_2') \xrightarrow{(u,\mathrm{req}),(z_1',\overline{\mathrm{req}}),(z_2',\overline{\mathrm{req}})} u,z_1',z_2' \vdash B_2(u,z_1',z_2') \tag{2}$$

$$z_i \vdash S_i(z_i) \xrightarrow{(z_i,\mathrm{req})} z_i \vdash S_i'(z_i) \tag{3}$$

The inference rules for bMSHR can be used to derive transition

$$u,z_1,z_2 \vdash B_2(u,z_1,z_2)|S_1(z_1)|S_2(z_2) \xrightarrow{(u,\mathrm{req})} u \vdash \nu z_1,z_2\ B_2(u,z_1,z_2)|S_1'(z_1)|S_2'(z_2)$$

a proof of which can be as follows. First, rule (par-b) is applied to productions (2) and (3) for S_1 and then applied again the production (3) for S_2. This yields a transition whose target graph is $u,z_1',z_2',z_1,z_2 \vdash B_2(u,z_1',z_2')|S_1(z_1)|S_2(z_2)$. Then, synchronisation is obtained by applying rule (merge-b) with substitution $\{z_1/z_1',z_2/z_2'\}$ so that, on node z_1 (resp. z_2), complementary actions $\overline{\mathrm{req}}$ by B_2 and req by S_1 (resp. S_2) are performed, producing a τ. Finally, z_1 and z_2 can be restricted using rule (res-b).

4 Milner SHR

This extension introduces a main feature of SHR, namely mobility. In the SHR framework mobility is intended as node mobility: nodes can be created and communicated together with actions, and when two actions interact corresponding nodes are merged. This allows to change the graph topology by creating new links during the computation.

We extend the definition of SHR transitions (Definition 3.1), adding the mobility part according to the approach of [10], which allows to send and merge both already existent and newly created nodes. We first formalise our alphabet of actions.

Definition 4.1 (Action signature). *An* action signature *is a triple* $(Act,\mathrm{ar},\varepsilon)$ *where* Act *is the set of actions,* $\varepsilon \in Act$*, and* $\mathrm{ar} : Act \to \mathbb{N}$ *is the arity function satisfying* $\mathrm{ar}(\varepsilon) = 0$.

The action signature $(Act_{Mil},\mathrm{ar},\varepsilon)$ for Milner synchronisation has further structure. In fact, $Act_{Mil} = \mathcal{A} \cup \overline{\mathcal{A}} \cup \{\tau,\varepsilon\}$ where $\mathcal{A}$ is the set of (input) actions and $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$ is the set of *co-actions*, τ is a special action with $\mathrm{ar}(\tau) = 0$. Finally, for each $a \in \mathcal{A}$ we have the constraint that $\mathrm{ar}(a) = \mathrm{ar}(\overline{a})$.

Mobility is modelled by letting function Λ in transitions to carry tuples of nodes. Hereafter, $\Lambda : \Gamma \to (Act \times \mathcal{N}^*)$ is a total function assigning, to each node $x \in \Gamma$, an action $a \in Act$ and a tuple $\mathbf{y}$ of node references sent to x such that $\mathrm{ar}(a) = |\mathbf{y}|$. We let $\mathrm{act}_\Lambda(x) = a$ and $\mathrm{n}_\Lambda(x) = \mathbf{y}$ when $\Lambda(x) = (a,\mathbf{y})$. Finally, the *set of communicated (resp. fresh) names* of Λ is $\mathrm{n}(\Lambda) = \{z \mid \exists x.z \in \mathrm{n}_\Lambda(x)\}$ (resp. $\Gamma_\Lambda = \mathrm{n}(\Lambda) \setminus \Gamma$).

Definition 4.2 (SHR transitions with mobility). *Given an action signature* $(Act, \mathrm{ar}, \varepsilon)$ *as described above, a* SHR transition with mobility *is a relation of the form:*

$$\Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$$

where $\pi : \Gamma \to \Gamma$ *is an idempotent renaming accounting for node merging such that* $\forall x \in \mathrm{n}(\Lambda).\ x\pi = x$. *Finally,* $\Phi = \Gamma\pi \cup \Gamma_\Lambda$.

As for σ in Definition 3.3, idempotency of π introduces equivalence classes on nodes and maps every node into a standard representative. By condition $\forall x \in \mathrm{n}(\Lambda).\ x\pi = x$, only references to representatives can be sent while $\Phi = \Gamma\pi \cup \Gamma_\Lambda$ states that free nodes are never erased ($\supseteq$) and new nodes are bound unless communicated ($\subseteq$).

Note that Φ is fully determined by Λ and π (since $\Gamma = \mathrm{dom}(\Lambda)$) and that, unlike in bMSHR, it might be $\Phi \neq \Gamma$.

The definition of productions is extended as follows.

Definition 4.3 (Productions). *A* production *is now an SHR transition of the form:*

$$x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n) \xrightarrow{\Lambda,\pi} \Phi \vdash G \tag{4}$$

where $\mathrm{rank}(L) = n$ *and* $x_1, \ldots, x_n$ *are all distinct. Production (4) is idle if* $\Lambda(x_i) = (\varepsilon, \langle\rangle)$ *for each i,* $\pi = id$ *and* $\Phi \vdash G = x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n)$.

As before, sets of productions include all the idle productions and are closed under injective renamings.

MSHR semantics (and the successive extensions) exploits a *most general unifier (mgu)* accounting for name fusions. The result of the application of the mgu is the fusion of nodes (new and old ones) changing the topology of graph (i.e. mobility).

The rules for MSHR presented below extend the ones for bMSHR with the machinery to deal with mobility.

Definition 4.4 (Inference rules for MSHR). *The admissible behaviours of MSHR are defined by the following inference rules.*

$$\textit{(par-M)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2 \qquad \Gamma' \vdash G_1' \xrightarrow{\Lambda',\pi'} \Phi' \vdash G_2' \qquad (\Gamma \cup \Phi) \cap (\Gamma' \cup \Phi') = \emptyset}{\Gamma, \Gamma' \vdash G_1 | G_1' \xrightarrow{\Lambda \cup \Lambda', \pi \cup \pi'} \Phi, \Phi' \vdash G_2 | G_2'}$$

$$\textit{(merge-M)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma\sigma \vdash G_1\sigma \xrightarrow{\Lambda',\pi'} \Phi' \vdash \nu U\ G_2\sigma\rho}$$

where $\sigma : \Gamma \to \Gamma$ *is an idempotent renaming and:*

1. *for all* $x, y \in \Gamma$ *such that* $x \neq y$, *if* $x\sigma = y\sigma \wedge \Lambda(x) \neq \varepsilon \wedge \Lambda(y) \neq \varepsilon$ *then* $(\forall z \in \Gamma \setminus \{x, y\}. z\sigma = x\sigma \Rightarrow \Lambda(z) = \varepsilon) \wedge \Lambda(x) = a \wedge \Lambda(y) = \overline{a} \wedge a \neq \tau$
2. $S_1 = \{\mathrm{n}_\Lambda(x) = \mathrm{n}_\Lambda(y) \mid x\sigma = y\sigma\}$
3. $S_2 = \{x = y \mid x\pi = y\pi\})$
4. $\rho = \mathrm{mgu}((S_1 \cup S_2)\sigma)$ *and* ρ *maps names to representatives in* $\Gamma\sigma$ *whenever possible*

5. $\Lambda'(z) = \begin{cases} (\tau, \langle\rangle) & \textit{if } x\sigma = y\sigma = z \wedge x \neq y \wedge \mathrm{act}_\Lambda(x) \neq \varepsilon \wedge \mathrm{act}_\Lambda(y) \neq \varepsilon \\ (\Lambda(x))\sigma\rho & \textit{if } x\sigma = z \wedge \mathrm{act}_\Lambda(x) \neq \varepsilon \\ (\varepsilon, \langle\rangle) & \textit{otherwise} \end{cases}$
6. $\pi' = \rho\lfloor_{\Gamma\sigma}$
7. $U = (\Phi\sigma\rho) \setminus \Phi'$

$$\textit{(res-M)} \quad \frac{\Gamma, x \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma \vdash \nu x\, G_1 \xrightarrow{\Lambda\lfloor_\Gamma, \pi\lfloor_\Gamma} \Phi' \vdash \nu Z\, G_2}$$

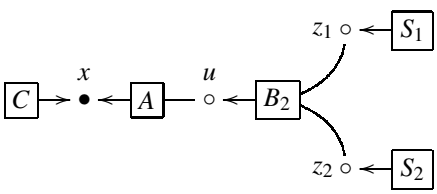
where:

6. $(\exists y \in \Gamma . x\pi = y\pi) \Rightarrow x\pi \neq x$
7. $\mathrm{act}_\Lambda(x) = \varepsilon \vee \mathrm{act}_\Lambda(x) = \tau$
8. $Z = \{x\}$ *if* $x \notin \mathrm{n}(\Lambda\lfloor_\Gamma), Z = \emptyset$ *otherwise*

$$\textit{(new-M)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2 \qquad x \notin \Gamma \cup \Phi}{\Gamma, x \vdash G_1 \xrightarrow{\Lambda \cup \{(x,\varepsilon,\langle\rangle)\},\pi} \Phi, x \vdash G_2}$$

Rules (par-M) and (new-M) are essentially as before. In rule (merge-M) now mobility must be handled. In particular, when actions and co-actions synchronise, parameters in corresponding positions are merged. This set of merges is computed in S_1 (condition 2), while S_2 (condition 3) describes old merges traced by π. Condition 4 combines the two sets of equations, updates them with σ and then chooses a representative for each equivalence class using a mgu. Among the possible equivalent mgus we choose one of those where nodes in $\Gamma\sigma$ are chosen as representatives (if they are in the equivalence class). This is necessary to avoid unexpected renamings of nodes because of fusions with new nodes which may then disappear. Note that (condition 5) Λ is updated with the merges specified by ρ and that (condition 6) π' is ρ restricted to the nodes of the graph which is the source of the transition. We may have to reintroduce restrictions (condition 7) if some nodes were extruded by the synchronised actions, since they will no more appear in the label. In rule (res-M) the bound node x must not be a representative if it belongs to a non trivial equivalence class.

Example 4.5. Consider the system in Example 3.4 with two servers S_1 and S_2, but where a client C must be first authenticated by an authority A. The graph representing the system is as follows:



We can model the fact that C is allowed to access the services by letting it move from node x to node u, namely by extruding the private node u to C. The productions for C and A are as follows:

$$x \vdash C(x) \xrightarrow{(x,\overline{\text{auth}},\langle y\rangle)} x,y \vdash C'(y) \qquad\qquad x,u \vdash A(x,u) \xrightarrow{(x,\text{auth},\langle u\rangle)} x,u \vdash A(x,u)$$

where, in the first production the client becomes attached to the received node y after the transition. In fact, when synchronisation is performed, new node y and node u are merged, with u as representative. Note that the restriction on u is reintroduced. Starting from $x \vdash \nu u\, C(x) \mid A(x,u)$ we will obtain $x \vdash \nu u\, C'(u) \mid A(x,u)$.

5 Synchronisation Algebras with Mobility

Synchronisation Algebras with Mobility (SAMs) allow us to parameterise SHR w.r.t. synchronisation models, e.g., MSHR will come out as just a particular instance of the general framework. SAMs extend synchronisation algebras (SAs), introduced in the framework of calculi for interaction such as CCS in [35]. Specifically, SAMs allow us to deal with mobility and to handle local resources (i.e., restriction), as they are used in SHR and more generally in mobile calculi. In general, SAMs must be able to express the synchronisation among any number of actions, each carrying its tuple of parameters. Actions from a multiset $\{|a_1,\ldots,a_n|\}$ can interact, and either they express compatible constraints, thus the system can perform a transition where these actions are executed on the same node, or they express incompatible constraints. For instance, in Milner synchronisation, a synchronisation among a, $\overline{a}$ and ε is allowed, while one involving a and b is not. With respect to [35], SAMs require to manage nodes carried by the actions.

A main ingredient in the formalisation of SAMs is the action synchronisation, which specifies an allowed pattern of interaction between two components. Before giving the definition, some notations are required.

Notation. The disjoint union of sets A and B is denoted as $A \uplus B$ and $\text{inj}_1 : A \to A \uplus B$ (resp. $\text{inj}_2 : B \to A \uplus B$) is the left (resp. right) inclusion. When no confusion arises, $\text{inj}_i(x)$ is written as x. Given $\text{inj}_i(x) \in A \uplus B$, $comp(\text{inj}_i(x))$ is element $\text{inj}_{3-i}(x)$ in $B \uplus A$. The set $\{1,\ldots,n\}$ is denoted by $\underline{n}$ (where $\underline{0} \stackrel{\text{def}}{=} \emptyset$) and id_n is the identity function on it. Finally, given two functions $f : A \to C$ and $g : B \to D$, $[f,g] : A \uplus B \to C \uplus D$ is the pairing of f and g, namely, $[f,g]$ applies f to elements in A and g to those in B.

Definition 5.1 (Action synchronisation). *Given an action signature* $\mathbf{A} = (Act, \text{ar}, \varepsilon)$, *an* action synchronisation *on* $\mathbf{A}$ *is a triple* $(a,b,(c,\text{Mob},\doteq))$ *where* $a,b,c \in Act$, $\text{Mob} : \underline{\text{ar}(c)} \to \underline{\text{ar}(a)} \uplus \underline{\text{ar}(b)}$ *and* $\doteq$ *is an equivalence relation on* $\underline{\text{ar}(a)} \uplus \underline{\text{ar}(b)}$.

An action synchronisation $(a,b,(c,\text{Mob},\doteq))$ relates two synchronising actions a and b to a triple $(c,\text{Mob},\doteq)$, representing the results of the synchronisation of a and b. Action c is the out-coming action, Mob is a communication function that tells how the parameters of c are taken from those of a and b and $\doteq$ is an equivalence relation on the parameters of a and b which generalises set S_1 in rule (merge-M) of Definition 4.4. Since actual parameters are not known at SAM-definition time, Mob and $\doteq$ are defined according to the positions of the parameters in the tuples: for instance

$\mathrm{Mob}(1) = \mathrm{inj}_2(3)$ means that the first parameter of c comes from the third parameter of the second action.

In order to finitely specify interactions among an unbound number of components, a compositional approach is needed. The intuition is that action synchronisation specifies how two components interact. The result of a synchronisation of many actions must be independent of the order of composition, hence composition of action synchronisations must be associative and commutative. The formalisation of this requirement is rather technical, thus we refer the interested reader to [22].

Action synchronisation relations impose conditions on action synchronisations.

Definition 5.2 (Action synchronisation relation). *An* action synchronisation relation *on an action signature* $\mathbf{A} = (Act, \mathrm{ar}, \varepsilon)$ *is a set ActSyn of action synchronisations s.t.:*

1. $(a, b, (c, \mathrm{Mob}, \doteq)) \in ActSyn \Rightarrow (c = \varepsilon \Leftrightarrow a = b = \varepsilon)$*;*
2. *composition of action synchronisations is associative and commutative.*

Condition 1 states that action ε can arise only as combination of actions ε. Note that condition 2 must be enforced not only as far as actions are concerned, but also for the part related to communication (Mob) and fusions ($\doteq$). It amounts to say that when all the actions in a tuple are composed, the result is independent on the order of composition. This can be formalised as a condition on the used SAM.

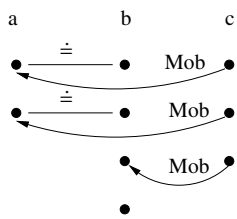
Having multiple action synchronisations for the same pair of interacting actions allows nondeterminism. In particular, the result of the synchronisation is nondeterministically chosen among the allowed alternatives.

As last step toward SAMs, we introduce a commonly used communication function and a related equivalence relation. The two definitions jointly define message passing, in the sense that they merge parameters in the same position and they make the result available as parameter of the composed action.

Definition 5.3 (Communication function for message passing). *The communication function for message passing* $MP_{i,j}$ *with* $i, j \in \mathbb{N}$ *is the function from* $\underline{\max(i, j)}$ *to (any superset of)* $\underline{i} \uplus \underline{j}$ *such that* $MP_{i,j}(m) = \mathrm{inj}_1(m)$ *if* $m \leq i$*,* $MP_{i,j}(m) = \overline{\mathrm{inj}_2(m)}$ *otherwise.*

Definition 5.4 (Equivalence relation for message passing). *The equivalence relation for message passing* EQ_i *with* $i \in \mathbb{N}$ *is the equivalence relation on any superset* S *of* $\underline{i} \uplus \underline{i}$ *given by* $id_S \cup \{(\mathrm{inj}_1(m), \mathrm{inj}_2(m)) \mid m \leq i\}$.

Example 5.5. A synchronisation between two actions a and b of arity 2 and 4, giving an action c of arity 3, with $\mathrm{Mob} = MP_{2,3}$ and $\doteq = EQ_2$ can be depicted as



The first two parameters of c are obtained by merging the ones from a and b, while the third one is taken from b. The fourth parameter of b is simply discarded.

Definition 5.6 (SAM). *A quintuple* $(\mathfrak{I}, \mathbf{A}, Init, Fin, ActSyn)$ *is a* Synchronisation Algebra with Mobility *over the action signature* $\mathbf{A}$ *where* $\mathfrak{I}$ *is an identifier,* $Init, Fin \subseteq Act$ *are the* initial actions *and* final actions *respectively and* $ActSyn$ *is an action synchronisation relation on* $\mathbf{A}$. *We require that* $\varepsilon \in Init$ *and*

1. $\forall i \in Init, a \in Act \setminus \{\varepsilon\}.(i, a, (c, \mathrm{Mob}, \doteq)) \in ActSyn \implies c = a \wedge \mathrm{Mob} = MP_{\mathrm{ar}(i),\mathrm{ar}(a)} \wedge \doteq \subseteq EQ_{\mathrm{ar}(a)}$;
2. $\forall a \in Act.\exists i \in Init.(i, a, (a, MP_{\mathrm{ar}(i),\mathrm{ar}(a)}, \doteq)) \in ActSyn$ *with* $\doteq \subseteq EQ_{\mathrm{ar}(a)}$.

Identifier $\mathfrak{I}$ is used to distinguish SAMs with the same structure but that can be composed in different ways (this will be used in SHR-HS, see § 7). Set $Init$ contains ε and some trivial actions that can be executed by nodes themselves, and they are a technical trick to deal with isolated nodes. Condition 1 specifies that the synchronisation of an initial action i with any action $a \neq \varepsilon$, if allowed, preserves a and its parameters. Condition 2 requires that each action a has an action i to synchronise with.

Finally, the set Fin of final actions contains the actions that are considered complete, and which thus do not require any further interaction in order to be meaningful. From a technical point of view, these are the actions allowed on bound channels, and they allow to deal with local resources.

Remark 5.7. From now on, to simplify the presentation, we will not write explicitly the action synchronisations obtained by commutativity; furthermore, given a SAM $A = (\mathfrak{I}, \mathbf{A}, Init, Fin, ActSyn)$, $(a, b, (c, \mathrm{Mob}, \doteq)) \in A$ denotes $(a, b, (c, \mathrm{Mob}, \doteq)) \in ActSyn$.

We present some examples of SAMs over a parametric set *inp* of input actions.

Definition 5.8 (Milner SAM). *For SAM* $Milner_{inp}$, $Init = \{\varepsilon\}$, $Fin = \{\tau, \varepsilon\}$ *where*

- $Act = \{\tau, \varepsilon\} \cup \bigcup_{a \in inp} \{a, \overline{a}\}$ *with* $\mathrm{ar}(\overline{a}) = \mathrm{ar}(a)$ *for each* $a \in inp$, $\mathrm{ar}(\tau) = 0$;
- $(\lambda, \varepsilon, (\lambda, MP_{\mathrm{ar}(\lambda),0}, EQ_0)) \in ActSyn$ *for each* $\lambda \in Act$,
  $(a, \overline{a}, (\tau, MP_{0,0}, EQ_{\mathrm{ar}(a)})) \in ActSyn$ *for each* $a \in inp$.

The first action synchronisation specifies that an action synchronising with ε is just propagated, together with its parameters. The second action synchronisation formalises the reaction of an action and the corresponding co-action. As expected, corresponding parameters are merged by $EQ_{\mathrm{ar}(a)}$.

Definition 5.9 (Hoare SAM). *SAM* $Hoare_{inp}$ *is given by:*

- $Act = Init = Fin = \{\varepsilon\} \cup inp$;
- $(\lambda, \lambda, (\lambda, MP_{\mathrm{ar}(\lambda),\mathrm{ar}(\lambda)}, EQ_{\mathrm{ar}(\lambda)})) \in ActSyn$ *for each* $\lambda \in Act$.

The only (schema of) action synchronisation in Hoare SAM models the agreement among the participants on the action to perform. During synchronisation corresponding parameters are merged and the results are propagated.

Definition 5.10 (Broadcast SAM). *For SAM Bdc_{inp}, $Init = \{\varepsilon\} \cup inp$, $Fin = \{\varepsilon\} \cup \bigcup_{a \in inp}\{\overline{a}\}$ and*

- $Act = \{\varepsilon\} \cup \bigcup_{a \in inp}\{a, \overline{a}\}$ *with* $\mathrm{ar}(\overline{a}) = \mathrm{ar}(a)$ *for each* $a \in inp$*;*
- $(a, \overline{a}, (\overline{a}, MP_{\mathrm{ar}(a),\mathrm{ar}(\overline{a})}, EQ_{\mathrm{ar}(a)})) \in ActSyn$ *for each* $a \in inp$*,*
  $(a, a, (a, MP_{\mathrm{ar}(a),\mathrm{ar}(a)}, EQ_{\mathrm{ar}(a)})) \in ActSyn$ *for each* $a \in inp \cup \{\varepsilon\}$*.*

The main difference w.r.t. Milner SAM is that here an output can synchronise with more than one input, thus when synchronisation is performed the result is the output itself, which can thus interact with further inputs. Notice also that two inputs can interact (this is required to ensure associativity), thus when an output is finally met, its parameters are merged with the ones of all the inputs. If no output is met then the resulting action is an input, which is not allowed on a bound channel. Also, broadcast SAM forces all the connected edges to interact with an output, in fact they cannot perform an action ε. Thus this SAM models secure broadcast, where a check is made to ensure that the broadcasted message is received by all the listeners. Multicast SAM Mul_{inp} can be easily obtained from Bdc_{inp} by adding $(\lambda, \varepsilon, (\lambda, MP_{\mathrm{ar}(\lambda),0}, EQ_0))$ to $ActSyn$, for each $\lambda \in Act$.

6 Parametric SHR

We outline *parametric SHR*, an SHR framework where the synchronisation policy can be freely chosen. The main ingredients of this model are a SAM, which specifies the synchronisation model used, and a set of inference rules, parametric on the above SAM, used to derive transitions from productions. Clearly, productions for parametric SHR must use actions in the set of actions Act_A of the SAM A used as parameter.

For space constraints we show just the rule (merge-p), and we outline the main differences between the other rules and the corresponding ones for Milner synchronisation. For a full formal account of the topic see [24,22].

Definition 6.1 (Merge rule for parametric SHR). *Let* $\sigma = \{x/y\}$*,*

$$\textit{(merge-p)} \quad \frac{\Gamma, x, y \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma, x \vdash G_1\sigma \xrightarrow{\Lambda',\pi'} \Phi' \vdash \nu U\ G_2\sigma\rho}$$

1. $\Lambda(x) = (a_1, \mathbf{v}_1), \Lambda(y) = (a_2, \mathbf{v}_2)$
2. $(a_1, a_2, (c, \mathrm{Mob}, \doteq)) \in ActSyn$
3. $S_1 = \{\mathbf{v}_{i_1}[j_1] = \mathbf{v}_{i_2}[j_2] \mid \mathrm{inj}_{i_1}(j_1) \doteq \mathrm{inj}_{i_2}(j_2)\}$
4. $S_2 = \{t = u \mid t\pi = u\pi\}$
5. $\rho = \mathrm{mgu}((S_1 \cup S_2)\sigma)$ *and* ρ *maps names to representatives in* Γ, x *whenever possible*
6. $\mathbf{w}[i] = (\mathbf{v}_j[k])\sigma\rho$ *if* $\mathrm{Mob}(i) = \mathrm{inj}_j(k)$
7. $\Lambda'(z) = \begin{cases} (c, \mathbf{w}) & \textit{if } z = x \\ (\mathrm{act}_\Lambda(z), (\mathrm{n}_\Lambda(z))\sigma\rho) & \textit{for each } z \in \Gamma \end{cases}$
8. $\pi' = \rho\restriction_{\Gamma,x}$
9. $U = \Phi\sigma\rho \setminus \Phi'$

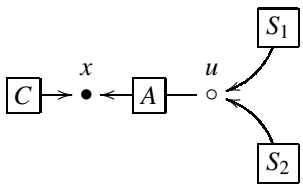
The main difference between the parametric inference rules and the ones in Definition 4.4 is that the parametric ones can be instantiated to model systems using a chosen synchronisation model.

To make the presentation clearer, rule (merge-p) uses a renaming $\sigma = \{x/y\}$ instead of a generic idempotent renaming. Synchronisation between two actions a_1 and a_2 is allowed iff there is an action synchronisation in *ActSyn* with a_1 and a_2 as first and second field respectively (condition 2). Also, the component Mob is used to compute the parameters of the resulting action (condition 6), while $\doteq$ is used to compute the first set of equalities (condition 3) which contributes to ρ.

In the rule for restriction, the action performed on the bound node must belong to Fin, while only actions in *Init* (with a tuple of fresh names as parameters) can be performed on a new node.

Parametric SHR fully recovers Milner SHR, in fact it is enough to instantiate it using the Milner SAM, see [24,22] for a formal statement. Naturally, parametric SHR can do more, as shown by the following example.

Example 6.2. We can exploit parametric SHR to improve the modelling of the system in Example 4.5. In fact, if we consider parametric SHR instantiated with Milner SAM, the example can be fully recovered. The synchronisation is obtained using rule (merge-p), which produces the same effect as the one in Example 4.5. Moreover, if we consider the SAM for broadcast synchronisation (Definition 5.10) instead of Milner, then the edge B_2, which is part of the infrastructure for communication, can be deleted. The graph for the new system with broadcast is:



In fact, broadcast synchronisation obtains the desired effect, by allowing an action $\overline{req}$ to interact with two actions *req*. The result of the broadcast synchronisation gives $\overline{req}$ that is in set Fin, thus u can be restricted.

7 SHR for Heterogeneous Systems

An *heterogeneous system* is a system where different subsystems exploit different synchronisation protocols. A further generalisation of SHR is *SHR for heterogeneous systems* (SHR-HS) [25,22] where heterogeneity is introduced by labelling nodes with SAMs that specify the synchronisation policy used on them. Hence, SHR-HS focuses on the management of the primitives available on nodes. Depending on circumstances, different strategies have to be followed. Specifically, at the network level the labelling is quite static, since it depends on hardware features, while at the application level it can change dynamically as a result of negotiations among different components. In fact, services that differ (e.g., w.r.t. their QoS aspects) can be conveniently described by different SAMs.

In SHR-HS this is modelled by allowing SAMs labelling a node to dynamically change as a result of a synchronisation among different parties. Technically, this corresponds to update the labelling when nodes are merged or created. Therefore, a set $\mathcal{A}\mathfrak{lg}$ of SAMs is assumed together with an operator $\diamond$ of SAM composition. Also, $\langle \mathcal{A}\mathfrak{lg}, \diamond, A_\varepsilon \rangle$ is assumed to be a commutative monoid. Associativity and commutativity are needed so that the result of the composition of SAMs does not depend on the order of composition. The requirement of having a neutral element is not restrictive since one can always add an unused element and set it as neutral element of the composition. A neutral element is useful when one wants to ensure that the label of a node x is preserved when x is merged with another node, e.g., with a parameter of an initial action. The main definitions of SHR are extended to deal with nodes labelled by SAMs, introduced by turning Γ into a function from nodes to SAMs.

Definition 7.1 (Labelled graphs). *A* labelled graph *is a judgement* $\Gamma \vdash G$ *where* Γ *is a finite function from* $\mathcal{N}$ *to* $\mathcal{A}\mathfrak{lg}$*; G is like before, but now restricted nodes are labelled, e.g.,* $\nu y : A.G$ *where* $y \in \mathcal{N}$ *and* $A \in \mathcal{A}\mathfrak{lg}$.

Extending previous notation, $x_1 : A_1, \ldots, x_n : A_n$ denotes a function mapping $x_i \in \mathcal{N}$ to $A_i \in \mathcal{A}\mathfrak{lg}$, for $i \in \{1, \ldots, n\}$. Structural congruence and isomorphisms of graphs are as in Definition 2.3 but, now, they must preserve SAMs labelling nodes.

Transitions $\Gamma \vdash G \xrightarrow{\Lambda,\pi} \Gamma \vdash G'$ are extended accordingly with the additional requirement that $\mathrm{act}_\Lambda(x) \in \Gamma(x)$. Moreover, productions $\Gamma \vdash L(\mathbf{x}) \xrightarrow{\Lambda,\pi} \Phi \vdash G$ impose some requirements on how the labels in the target graph are chosen. Any SAM can be used to label nodes not in $\mathrm{dom}(\Gamma)$, i.e., generated in the production, while for a node $x \in \mathrm{dom}(\Gamma)$, $\Phi(x\pi)$ is $\Gamma(x_1) \diamond \ldots \diamond \Gamma(x_n)$ where $x_1, \ldots, x_n$ are all the nodes that π maps to $x\pi$.

In the inference rules, a production can be applied to an edge only if it specifies correct labels for the attached nodes. To specify SAMs applicable in different circumstances, suitable meta-notations can be used. Moreover, since now contexts are functions, both their domains (i.e., the sets of nodes) and their labelling SAMs must be kept into account. As an example, we give the merge rule (the others can straightforwardly be adapted from rules in Definition 4.4 and can be found in [25,22]).

Definition 7.2 (Merge rule for SHR-HS). *Let* $\sigma = \{x/y\}$,

$$\textit{(merge-HS)} \quad \frac{\Gamma, x : A, y : A \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma, x : A \vdash G_1\sigma \xrightarrow{\Lambda',\pi'} \Phi' \vdash \nu U\ G_2\sigma\rho}$$

1. $\Lambda(x) = (a_1, \mathbf{v}_1), \Lambda(y) = (a_2, \mathbf{v}_2)$
2. $(a_1, a_2, (c, \mathrm{Mob}, \doteq)) \in A$
3. $S_1 = \{\mathbf{v}_{i_1}[j_1] = \mathbf{v}_{i_2}[j_2] \mid \mathrm{inj}_{i_1}(j_1) \doteq \mathrm{inj}_{i_2}(j_2)\}$
4. $S_2 = \{t = u \mid t\pi = u\pi\}$
5. $\rho = \mathrm{mgu}((S_1 \cup S_2)\sigma)$ *and* ρ *maps names to representatives in* $\mathrm{dom}(\Gamma) \cup \{x\}$ *whenever possible*
6. $\mathbf{w}[i] = (\mathbf{v}_j[k])\sigma\rho$ *if* $\mathrm{Mob}(i) = \mathrm{inj}_j(k)$
7. $\Lambda'(z) = \begin{cases} (c, \mathbf{w}) \textit{ if } z = x \\ (\mathrm{act}_\Lambda(z), (\mathrm{n}_\Lambda(z))\sigma\rho) \textit{ for each } z \in \Gamma \end{cases}$

8. $\pi' = \rho\downarrow_{\mathrm{dom}(\Gamma)\cup\{x\}}$
9. $\mathrm{dom}(U) = \mathrm{dom}(\Phi)\sigma\rho \setminus \mathrm{dom}(\Phi')$
10. *the label of each node* $x \in \mathrm{dom}(U) \cup \mathrm{dom}(\Phi')$ *is computed as follows:* x *is the representative according to* $\sigma\rho$ *of an equivalence class* $\{x_1,\ldots,x_n\}$ *of nodes which have in* Φ *labels* $A_1,\ldots,A_n$*. Then the label of* x *is* $A_1 \diamond \ldots \diamond A_n$

Nodes x and y can be merged only if they have the same label A, and the interaction is performed according to one of the action synchronisations in its action synchronisation relation. The node resulting from the merge of x and y is also labelled with A. Nodes not involved in the merging preserve their label while the others get their labels as resulting from the application of $\diamond$.

Example 7.3. The system of Example 6.2 can be now more accurately modelled by simultaneously using a SAM for Milner synchronisation on actions for authorisation, and one for broadcast of requests. Thus on each node only the desired actions are available. This avoids undesired executions caused by malicious clients. Available synchronisations are exploited by the authority to ensure that clients can issue only authorised requests. Also, actions can specify the synchronisation policy (e.g, Milner or broadcast synchronisation) so that clients dynamically choose what protocol to use.

At a first sight, it might be argued that parametric SHR can model heterogeneous systems. However, parametric SHR does not fit with heterogeneous systems because it makes each synchronisation policy available on each node, which is not what heterogeneous systems (as we consider them here) require. On the other hand, parametric SHR is a special case of SHR-HS where a unique SAM is used (as shown in [22]).

8 SHReQ: Coordinating Application Level QoS

Awareness of *Quality of Service* (QoS) is an emergent exigency in SOC which is no longer considered only as a low-level aspect of systems. The ability of formally specifying and programming QoS requirements may represent a significant added-value of the SOC paradigm. Moreover, QoS information can drive the design and development of programming interfaces and languages for QoS-aware middlewares as well as to drive the search-bind cycle of SOC.

In SHReQ, a calculus based on SHR, abstract high-level QoS requirements are expressed as *constraint-semiring* [1] and embedded in the rewriting mechanism which is parameterised with respect to a given c-semiring. Basically, values of c-semirings are synchronisation actions so that synchronising corresponds to the product operation of c-semirings that can be regarded as the simultaneous satisfaction of the QoS requirements of the participants to the synchronisation.

Definition 8.1 (C-semiring). *An algebraic structure* $\langle S,+,\cdot,\mathbf{0},\mathbf{1}\rangle$ *is a* constraint semiring *if* S *is a set with* $\mathbf{0},\mathbf{1} \in S$*, and* $+$ *and* $\cdot$ *are binary operations on* S *such that:*

- $+$ *is commutative, associative, idempotent,* $\mathbf{0}$ *is its unit element and* $\mathbf{1}$ *is its absorbing element (i.e.,* $a+\mathbf{1}=\mathbf{1}$*, for any* $a \in S$*);*
- $\cdot$ *is commutative, associative, distributes over* $+$*,* $\mathbf{1}$ *is its unit element, and* $\mathbf{0}$ *is its absorbing element (i.e.,* $a\cdot\mathbf{0}=\mathbf{0}$*, for any* $a \in S$*).*

The additive operation $(+)$ of a c-semiring induces a partial order on S defined as $a \leq_S b \iff \exists c : a + c = b$. The minimum is thus $\mathbf{0}$ and the maximum is $\mathbf{1}$. C-semirings have two distinguished features that result very useful for modelling abstract QoS. First, the cartesian product of c-semirings is still a c-semiring, hence we can uniformly deal with many different quantities simultaneously. Second, partial order $\leq_S$ provides a mechanism of choice. These features make c-semirings suitable for reasoning about multi-criteria QoS issues [6,7]. The fact that c-semiring structure is preserved by cartesian product is here exploited to compose synchronisation policies.

Example 8.2. The following examples introduce some c-semirings together with their intended application to model QoS attributes. A more complete list can be found in [1].

- The boolean c-semiring $\langle \{true, false\}, \vee, \wedge, false, true \rangle$ can be used to model network and service availability.
- The optimisation c-semiring $\langle \text{Real}, min, +, +\infty, 0 \rangle$ applies to a wide range of cases, like prices or propagation delay.
- The max/min c-semiring $\langle \text{Real}, max, min, 0, +\infty \rangle$ can be used to formalise bandwidth, while the corresponding c-semiring over the naturals $\langle \mathbb{N}, max, min, 0, +\infty \rangle$ can be applied for resource availability.
- Performance can be represented by the probabilistic c-semiring $\langle [0,1], max, \cdot, 0, 1 \rangle$.
- Security degrees are modelled via the c-semiring $\langle [0,1,\ldots,n], max, min, 0, n \rangle$, where n is the maximal security level (unknown) and 0 is the minimal one (public).

Hereafter, given a c-semiring $\langle S, +, \cdot, \mathbf{0}, \mathbf{1} \rangle$, $\text{ar}_S : S \to \mathbb{N}$ is an arity function assigning arities to values in S. Graphs in SHReQ are called *weighted graphs* because values is S are used as weights and record quantitative information on the computation of the system.

Syntactically, SHReQ graphs are as those in SHR-HS where SAMs are replaced by c-semiring values. We write $x_1 : s_1, \ldots, x_n : s_n \vdash G$ for the weighted graph whose weighting function maps x_i to s_i, for $i \in \{1, \ldots, n\}$.

SHReQ rewriting mechanism relies on c-semirings where additional structure is defined. More precisely, we assume sets *Sync*, *Fin* and *NoSync* such that

- $Sync \subseteq Fin \subseteq S$, $\mathbf{1} \in Sync$ and $\text{ar}_S(s) = 0$ if $s \in Sync$;
- $NoSync \subseteq S \setminus Fin$, $\mathbf{0} \in NoSync$ and $\forall s \in S. \forall t \in NoSync. s \cdot t \in NoSync$.

The intuition follows the SAM approach (Definition 5.6) and it is that *Fin* contains those values of S representing events of complete synchronisations. Among the actions in *Fin* we can select a subset of "pure" synchronisation actions, namely complete synchronisations that do not expose nodes. Set *NoSync*, on the contrary, contains the values that represent "impossible" synchronisations.

SHReQ productions follow the lines of Definition 4.3 and 4.4, but have a slightly different interpretation. For simplicity, we avoid the π component in SHReQ transitions and require that free nodes cannot be merged. Technically, this is obtained by considering undefined the most general unifier operation when it yields the fusion of two free nodes. In [18] the general unification is defined for SHReQ.

Definition 8.3 (SHReQ productions). *Let S be a c-semiring $\langle S,+,\cdot,\mathbf{0},\mathbf{1}\rangle$. A* SHReQ production *is a production*

$$\Gamma \vdash L(x_1,\ldots,x_n) \xrightarrow{\Lambda} \Phi \vdash G \tag{5}$$

built on top of the action signature $(S,\mathrm{ar}_S,\mathbf{1})$ where Γ maps nodes in $\{x_1,\ldots,x_n\}$ to S.

Production (5) states that, in order to replace L with G in a graph H, applicability conditions expressed by the function Γ on the attachment nodes of L must be satisfied in H and, henceforth, L "contributes" to the rewriting by offering Λ in the synchronisation with adjacent edges. Function Γ expresses the *minimal* QoS requirements on the environment in order to apply the production, i.e., given $x \in \mathrm{dom}(\Gamma)$, the weight w on the node corresponding to x must satisfy $\Gamma(x) \leq w$. As before, function Φ is fully determined by Γ and Λ, where the weight of new nodes is set to $\mathbf{1}$ (i.e., $\Phi(y) = \mathbf{1}$ if $y \in \Gamma_\Lambda$), while for old nodes it traces the result of the synchronisation performed on them.

In production (5), c-semiring values play different roles in Γ and Λ: in Γ, they are interpreted as the minimal requirements to be fulfilled by the environment; in Λ they are the "contribution" that L yields to the synchronisation with the surrounding edges.

For space limitations, we only give the inference rule (merge-s) for merging nodes, the other rules being a simple rephrasing of those seen in previous sections. Rule (merge-s) is an adaptation of (merge-p) in Definition 6.1:

$$\text{(merge-s)} \quad \frac{\Gamma, x : r, y : s \vdash G_1 \xrightarrow{\Lambda \cup \{(x,s_1,\mathbf{v}_1),(y,s_2,\mathbf{v}_2)\}} \Phi \vdash G_2}{\Gamma, x : r+s \vdash G_1\sigma \xrightarrow{\Lambda'} \Phi' \vdash \nu U\ G_2\sigma\rho}$$

with $\sigma = \{x/y\}$ and Λ', Φ', ρ and U computed as in Definition 6.1, where action synchronisation on x is given by the c-semiring multiplication and its result is saved as the new weight of the synchronising node (i.e., $x : s_1 \cdot s_2$) both for free nodes and for nodes in U. In order to ensure applicability of productions also when there are more resources available than required, the following rule is introduced.

$$\text{(order-s)} \quad \frac{\Gamma, x : r \vdash G_1 \xrightarrow{\Lambda} \Phi \vdash G_2 \qquad r \leq t}{\Gamma, x : t \vdash G_1 \xrightarrow{\Lambda} \Phi \vdash G_2}$$

The other rules are similar to the ones in Definition 4.4.

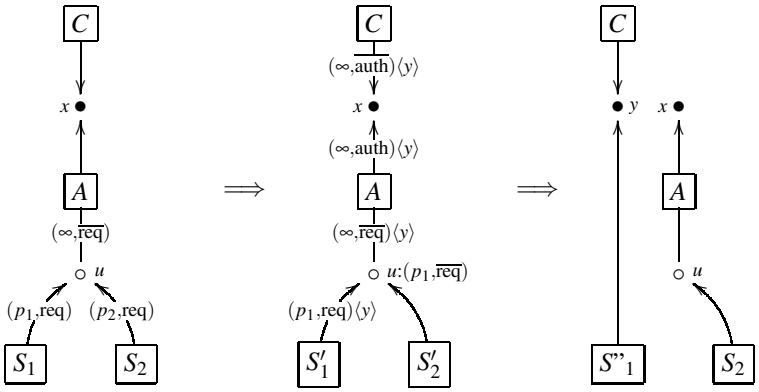
Example 8.4. Let us consider Example 6.2. We can model the authority choosing the server that offers the cheapest service. To this aim, we use the cartesian product of two c-semirings. The first c-semiring is: $\langle R^+, max, min, 0, \infty\rangle$, for the price of the service. The second c-semiring is used for synchronisation. In this way, we are able to define a general synchronisation policy as a unique c-semiring combining a classical synchronisation algebra with QoS requirements. The second c-semiring corresponds to multicast synchronisation. Assume $W = \{\mathrm{req}, \mathrm{auth}, \overline{\mathrm{req}}, \overline{\mathrm{auth}}, \mathbf{1}_W, \mathbf{0}_W, \bot\}$. Set W can be equipped with a c-semiring structure $\langle W,+,\cdot,\mathbf{0}_W,\mathbf{1}_W\rangle$, where:

$$\mathrm{req}\cdot\overline{\mathrm{req}} = \overline{\mathrm{req}}, \quad \mathrm{auth}\cdot\overline{\mathrm{auth}} = \overline{\mathrm{auth}}, \quad \mathrm{req}\cdot\mathrm{req} = \mathrm{req}, \quad \mathrm{auth}\cdot\mathrm{auth} = \mathrm{auth},$$
$$a,b \in W \setminus \{\mathbf{0}_W, \mathbf{1}_W\} \wedge a \neq b \wedge b \neq \overline{a} \implies a \cdot b = \bot$$

plus rules obtained by commutativity and the ones for $\mathbf{0}_W$ and $\mathbf{1}_W$.

The operation $+$ is obtained by extending the c-semiring axioms for the additive operation with $a+a=a$ and $a,b \notin \{\mathbf{0}_W, \mathbf{1}_W\} \wedge a \neq b \implies a+b=\bot$, for all $a,b \in W$.

Below we show a graphical representation of a two steps derivation. Instead of reporting productions for each rewriting step, tentacles are decorated with actions. For the sake of clarity, in each step we only write actions and weights of the relevant nodes.



The first step selects the server with the lowest price where p_i is the price for S_i (in this step no names are communicated). This is obtained as the result of the synchronisation in u, i.e., $((\overline{\text{req}} \cdot \text{req}) \cdot \text{req},\ min(\infty, p_1, p_2))$. Assuming p_1 less than p_2 the new weight of u is $(\overline{\text{req}}, p_1)$. The second step shows the client connecting to the cheapest server S_1 (informed by A) by connecting to a new node y. After the first synchronisation, the cheapest server is identified by the authority using the new weight on node u. This guides the behaviour of S_1 and of the authority to produce the new connection to the client. In particular, the applicability condition of server rule requires its price to be less than or equal to the price on the node, and this can be satisfied only by the cheapest one (we suppose for simplicity that server costs are unique).

9 Concluding Remarks

In this tutorial paper we introduced SHR as a basic metalanguage with strong theoretical foundations for describing distributed systems within the SOC paradigm. We have addressed the key issues of the SHR model describing features like mobility, heterogeneity and Quality of Service. A great deal of future work remains. At the experimental level, more experience in specifying and designing service oriented applications is needed. The problem of supporting the development of highly decentralised applications (from requirement and design to implementation and maintenance) is at the edge of research in software engineering. Indeed, software engineering technologies must support the shift from the client-server interaction model to other models which better accommodate the constraints posed by the SOC paradigm. We argue that the SHR model fosters a declarative approach by identifying the interaction borders of services where satisfaction of certain properties (e.g. Quality of Service) has a strong impact on the behaviours. Some preliminary results on the exploitation of the SHR model in specifying and designing internetworking systems can be found in [11]. In this perspective the development of

tool support for the SHR framework would be of great value. In the short term, we plan to experiment our framework to model workflow among services (e.g. by extending the Petri Nets translation developed in [34]).

At the foundational level, future work will be focused on the definition of abstract semantics for the SHR model. A basic question is "what is the appropriate notion of semantic equivalence for SHR?". Bisimulation-based equivalences have been proved to be a powerful basis for semantic equivalence of process calculi. Bisimulation semantics has the main advantage of capturing the idea of interaction within arbitrary contexts thus providing the semantic machinery for compositional reasoning. Hence, a main problem is understanding whether bisimilarity is a congruence w.r.t. the operators of system composition or not, i.e. whether the compositions of bisimilar systems are bisimilar or not. If they are, then the observational properties of a complex system can be derived by composing the results obtained on their components. The development of a compositional bisimulation semantics for SHR is not straightforward, since it requires to define a suitable algebra of graphs and exploit bialgebraic techniques in a non trivial way. Some preliminary results can be found in [22]. A further line of future research concerns the development of a "true concurrent" semantics for SHR where the notions of causality and independence are explicitly represented. The ability of reasoning on the causality flow could be particularly useful to manage the complexity of service oriented applications. For instance the analysis of service workflows can benefit from knowledge about causality: it suffices to focus on the causal dependencies among service invocations to understand the properties of the business interactions. We plan to extend the techniques introduced in [9] to equip SHR with a truly concurrent semantics.

Acknowledgements. The authors thank the anonymous reviewers for their valuable comments and suggestions.

References

1. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, 1997.
2. L. Cardelli and A. D. Gordon. Mobile ambients. In *Proc. of FoSSaCS'98*, volume 1378 of *LNCS*, pages 140–155. Springer, 1998.
3. I. Castellani and U. Montanari. Graph grammars for distributed systems. In *Graph-Grammars and Their Application to Computer Science*, volume 153 of *LNCS*, pages 20–38. Springer, 1983.
4. A. Corradini, P. Degano, and U. Montanari. Specifying highly concurrent data structure manipulation. In *Proc. of Computing 85*. Elsevier Science, 1985.
5. A. Corradini and D. Hirsch. An operational semantics of CommUnity based on graph transformation systems. In *Proc. of GT-VMT 2004*, volume 109 of *Elect. Notes in Th. Comput. Sci.*, pages 111–124. Elsevier Science, 2004.
6. R. De Nicola, G. Ferrari, U. Montanari, R. Pugliese, and E. Tuosto. A Formal Basis for Reasoning on Programmable QoS. In *International Symposium on Verification – Theory and Practice*, volume 2772 of *LNCS*, pages 436–479. Springer, 2003.
7. R. De Nicola, G. Ferrari, U. Montanari, R. Pugliese, and E. Tuosto. A process calculus for qos-aware applications. In *Proc. of Coordination'05*, volume 3454 of *LNCS*, pages 33–48. Springer, 2003.

8. R. De Nicola, G. Ferrari, and R. Pugliese. KLAIM: A kernel language for agents interaction and mobility. *IEEE Trans. Software Eng.*, 24(5):315–330, 1998.
9. P. Degano and U. Montanari. A model for distributed systems based on graph rewriting. *Journal of the ACM*, 34(2):411–449, 1987.
10. G. Ferrari, U. Montanari, and E. Tuosto. A LTS semantics of ambients via graph synchronization with mobility. In *ICTCS'01*, volume 2202 of *LNCS*, pages 1–16. Springer, 2001.
11. G. Ferrari, U. Montanari, and E. Tuosto. Graph-based models of internetworking systems. In *Formal Methods at the Crossroads: From Panacea to Foundational Support*, volume 2757 of *LNCS*, pages 242–266. Springer, 2003.
12. C. Fournet and G. Gonthier. The reflexive CHAM and the join-calculus. In *Proc. of POPL '96*, pages 372–385, 1996.
13. F. Gadducci and U. Montanari. The tile model. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.
14. D. Hirsch. *Graph Transformation Models for Software Architecture Styles*. PhD thesis, Departamento de Computación, Facultad de Ciencias Exactas y Naturales, U.B.A., 2003.
15. D. Hirsch, P. Inverardi, and U. Montanari. Reconfiguration of software architecture styles with name mobility. In *Proc. of Coordination '00*, volume 1906 of *LNCS*, 2000.
16. D. Hirsch and U. Montanari. Synchronized hyperedge replacement with name mobility. In *Proc. of CONCUR'01*, volume 2154 of *LNCS*. Springer, 2001.
17. D. Hirsch and E. Tuosto. SHReQ: A framework for coordinating application level QoS. In *Proc. of SEFM'05*, pages 425–434. IEEE Computer Society Press, 2005.
18. D. Hirsch and E. Tuosto. Coordinating Application Level QoS with SHReQ. *Journal of Software and Systems Modelling*, 2006. Submitted.
19. C. A. R. Hoare. A model for communicating sequential processes. In *On the Construction of Programs*. Cambridge University Press, 1980.
20. B. König and U. Montanari. Observational equivalence for synchronized graph rewriting. In *Proc. of TACS'01*, volume 2215 of *LNCS*, pages 145–164. Springer, 2001.
21. I. Lanese. Exploiting user-definable synchronizations in graph transformation. In *Proc. of GT-VMT'06*, Elect. Notes in Th. Comput. Sci. ES, 2006. To appear.
22. I. Lanese. *Synchronization Strategies for Global Computing Models*. PhD thesis, Computer Science Department, University of Pisa, Pisa, Italy, 2006. Forthcoming.
23. I. Lanese and U. Montanari. A graphical fusion calculus. In *Proceedings of the Workshop of the COMETA Project on Computational Metamodels*, volume 104 of *Elect. Notes in Th. Comput. Sci.*, pages 199–215. Elsevier Science, 2004.
24. I. Lanese and U. Montanari. Synchronization algebras with mobility for graph transformations. In *Proc. of FGUC'04 – Foundations of Global Ubiquitous Computing*, volume 138 of *Elect. Notes in Th. Comput. Sci.*, pages 43–60. Elsevier Science, 2004.
25. I. Lanese and E. Tuosto. Synchronized hyperedge replacement for heterogeneous systems. In *Proc. of Coordination'05*, volume 3454 of *LNCS*, pages 220–235. Springer, 2005.
26. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1982.
27. R. Milner, J. Parrow, and J. Walker. A calculus of mobile processes, I and II. *Inform. and Comput.*, 100(1):1–40,41–77, 1992.
28. J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proc. of LICS'98*, pages 176–185. IEEE Computer Society Press, 1998.
29. G. D. Plotkin. A structural approach to operational semantics. *J. Log. Algebr. Program.*, 60-61:17–139, 2004.
30. J. Riely and M. Hennessy. Distributed processes and location failures. *TCS*, 266(1–2):693–735, 2001.

31. F. Rossi and U. Montanari. Graph rewriting, constraint solving and tiles for coordinating distributed systems. *Applied Categorical Structures*, 7(4):333–370, 1999.
32. G. Rozenberg, editor. *Handbook of graph grammars and computing by graph transformations, vol. 1: Foundations*. World Scientific, 1997.
33. E. Tuosto. *Non-Functional Aspects of Wide Area Network Programming*. PhD thesis, Computer Science Department, University of Pisa, Italy, 2003.
34. W. M. P. van der Aalst and K. B. Lassen. Translating workflow nets to BPEL4WS. Technical Report WP 145, Eindhoven University of Technology, 2005.
35. G. Winskel. Synchronization trees. *TCS*, 34:33–82, 1984.