Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 138 (2005) 43–60
www.elsevier.com/locate/entcs

Synchronization Algebras with Mobility for Graph Transformations ⋆

Ivan Lanese and Ugo Montanari[1] ,[2]

Dipartimento di Informatica
Università di Pisa
Pisa, Italy

Abstract

We propose a generalization of synchronization algebras that allows to deal with mobility and local resource handling. We show how it can be used to model communication primitives for distributed and mobile computations, such as the ones used in the global computing area. We propose a graph transformation formalism in the Synchronized Hyperedge Replacement approach which is parametric w.r.t. the synchronization algebra and thus allows to model complex systems based on the chosen communication primitives. We thus unify different models described in the literature and we allow to easily define new ones. We present various examples and a case study on Fusion Calculus, showing how different semantics for it can be derived using different synchronization algebras.

Keywords: graph transformation, Synchronized Hyperedge Replacement, synchronization algebra, mobility, Fusion Calculus.

1 Introduction

Global computing (GC) deals with huge computational systems which are deployed on world-scale areas. In order to develop, analyze and manage such a complex kind of systems, where different issues like coordination, security and mobility interact, suitable formal tools are required.

⋆ Research supported in part by EU-FET project **AGILE** IST-2001-32747.
[1] Email: lanese@di.unipi.it
[2] Email: ugo@di.unipi.it

doi:10.1016/j.entcs.2005.05.004

We argue that these systems must be analyzed at different levels of abstraction. In particular, at the low level, protocols and algorithms must be developed to build an efficient and reliable middleware out of basic infrastructures which are usually asynchronous and unreliable. At the higher level, the primitives that are defined in the middleware must be used to coordinate the evolution of the different subsystems. In the GC scenario we may have also complex synchronization primitives involving multiple computational entities, thus we need a suitable way to model them, and then we need a framework to describe large systems based on these primitives.

In order to make the implementation step easier, computation must be distributed, and the coordination part must be clearly individuate. In particular, computations that require a strong knowledge on the topology of the whole system must be disallowed, since this knowledge is usually not available.

We choose graphs (or, more precisely, hypergraphs) as basic model, since they have a suggestive representation and they naturally model the topology of distributed systems. We associate to graphs the following computational interpretation: edges represent processes or subsystems, while nodes model channels or ports. Communication is performed via shared nodes.

Computation is modelled by graph transformations in the Synchronized Hyperedge Replacement (SHR) approach [2]. In this framework, context-free productions describe the local evolution of an edge. Such a kind of productions can easily be implemented since it involves just one component. In order to specify complex reconfigurations, we introduce a synchronization mechanism based on constraints imposed on shared nodes to decide which productions can be applied in a single step. We also want to model mobility directly, and we use name mobility in the Fusion Calculus [14] style to do that [7,4].

The approach based on local productions differentiates SHR from other graph transformation frameworks such as Double Pushout [3] and Bigraphs [5], where rules with arbitrarily complex left-hand side are used, and it allows a distributed implementation [2].

We propose a framework where the synchronization and the mobility patterns are specified by a suitable algebra, which extends synchronization algebras [15]. With respect to standard synchronization algebras our approach is also able to deal with mobility and local resource handling, which are two essential features of GC systems.

Synchronization algebras with mobility allow both to recover the synchronization models presented in the literature for SHR [4,12,9] and to easily define more complex ones. This is important since the models in the literature are quite low level, while complex high level primitives are useful for GC.

We present many examples to show both how synchronization algebras

with mobility can be used to specify synchronization primitives and how SHR can be used to model scenarios of interest for global and ubiquitous computing.

As an interesting case study we present a mapping from Fusion Calculus [14] (a process algebra that extends π-calculus [13] with the concept of fusion) into SHR. As proved in [11] an operational correspondence exists between Fusion Calculus and SHR with Milner synchronization. We show that using synchronization algebras different semantics can be provided for Fusion Calculus, thus allowing to apply it to systems based on different communication primitives.

Structure of the paper

The background on graphs, Synchronized Hyperedge Replacement and synchronization algebras is presented in §2. In §3 we present synchronization algebras with mobility while §4 contains the rules for parametric SHR. Some examples are shown in §5 and a case study on Fusion Calculus is discussed in §6. Finally §7 presents conclusions and traces for future work.

2 Background

2.1 Graphs as syntactic judgements

We want to model systems using hypergraphs, which generalize graphs allowing (hyper)edges to be connected to any number of attachment nodes. In particular, we use (hyper)graphs with labelled edges, that is an edge is an atomic item with a label (from a ranked alphabet $LE = \{LE_n\}_{n=0,1,\ldots}$) and with as many ordered tentacles as the rank rank(L) of its label L. A set of nodes, together with a set of such edges, forms a graph if each edge is connected, by its tentacles, to its attachment nodes. A graph is connected to its environment by an interface which is a subset of its nodes. Nodes in the interface are called free nodes, while other nodes are said bound. We consider graphs up to isomorphisms that preserve free nodes, labels of edges, and connections between edges and nodes.

We use a linear representation for graphs as (syntactic) judgements which is more suitable for defining transformations [8]. In this representation nodes correspond to names, free nodes to free names and edges to basic terms of the form $L(x_1, \ldots, x_n)$, where x_i are arbitrary names and $L \in LE_n$. We use a constant *nil* to represent the empty graph, a parallel composition operator $|$ to build large graphs from smaller ones and a ν operator to bind nodes.

Definition 2.1 (Graphs as judgements) *Let $\mathcal{N}$ be a fixed infinite set of names and LE a ranked alphabet of labels. A judgement is of the form $\Gamma \vdash G$*

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

Table 1
Structural congruence for graph terms.

where:

(i) $\Gamma \subseteq \mathcal{N}$ *is a finite set of names (the free nodes of the graph);*

(ii) G *is a term generated by the grammar*
$G ::= L(\boldsymbol{x}) \mid G|G \mid \nu y\ G \mid nil$
where $\boldsymbol{x}$ *is a vector of names,* L *is an edge label with* $\text{rank}(L) = |\boldsymbol{x}|$ *and* y *is a name.*

We define the restriction operator ν *as a binder. We denote with* fn *the function that given a term* G *returns the set* $\text{fn}(G)$ *of free names in* G*. We demand that* $\text{fn}(G) \subseteq \Gamma$.

When defining the interfaces, we use the notation Γ, x to denote the set obtained by adding x to Γ, assuming $x \notin \Gamma$ and Γ_1, Γ_2 to denote the union of Γ_1 and Γ_2, assuming $\Gamma_1 \cap \Gamma_2 = \emptyset$.

Graph terms are considered up to the axioms of structural congruence in Table 1. As far as judgements are concerned, we define $\Gamma \vdash G \equiv \Gamma' \vdash G'$ iff $\Gamma = \Gamma'$ and $G \equiv G'$.

Axioms (AG1), (AG2) and (AG3) define respectively the associativity, commutativity and identity over *nil* for operation $|$. Axioms (AG4) and (AG5) state that nodes of a graph can be hidden 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.

Theorem 2.2 (Soundness of the representation [6]) *Judgements up to structural congruence are isomorphic to graphs up to isomorphisms.*

2.2 Synchronized Hyperedge Replacement

We introduce here Synchronized Hyperedge Replacement (SHR) [2,7,4], a graph transformation formalism where transitions are specified by synchronizing context-free productions which describe how single edges are rewritten. A production rewrites an edge into a graph with the same interface, and it exposes actions on the nodes of its interface. Actions are constraints, and a set of productions can be applied in one step if the constraints imposed on

shared nodes are compatible. Tuples of references to nodes are attached to actions, and represent names that are communicated during the synchronization. Furthermore productions can force merges among nodes in the interface via a suitable substitution.

We use a notation based on judgements also for transitions.

Definition 2.3 (SHR transition) *Let Act be a set of actions, and given $a \in Act$ let* $\mathrm{ar}(a)$ *be its arity. A SHR transition is of the form:*

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

where $\Gamma \vdash G$ and $\Phi \vdash G'$ are judgements for graphs, $\Lambda : \Gamma \to (Act \times \mathcal{N}^*)$ is a total function and $\pi : \Gamma \to \Gamma$ is an idempotent substitution. Function Λ assigns to each node x the action $a \in Act$ and the vector $\boldsymbol{y}$ of node references exposed on x by the transition (in a more message-passing view, we say that node references are sent to x). If $\Lambda(x) = (a, \boldsymbol{y})$ then we define $\mathrm{act}_\Lambda(x) = a$ *and* $\mathrm{n}_\Lambda(x) = \boldsymbol{y}$. *We require that* $\mathrm{ar}(\mathrm{act}_\Lambda(x)) = |\,\mathrm{n}_\Lambda(x)|$.

We define:

- $\mathrm{n}(\Lambda) = \{z | \exists x.z \in \mathrm{n}_\Lambda(x)\}$ *set of exposed names;*
- $\Gamma_\Lambda = \mathrm{n}(\Lambda) \setminus \Gamma$ *set of fresh names that are exposed.*

Substitution π allows to merge nodes. Since π is idempotent, it maps every node into a standard representative of its equivalence class. We require that $\forall x \in \mathrm{n}(\Lambda).\pi(x) = x$, i.e. only references to representatives can be exposed. Furthermore we require $\Phi = \pi(\Gamma) \cup \Gamma_\Lambda$, namely free nodes are never erased ($\supseteq$) and new nodes are bound unless exposed ($\subseteq$).

Note that the set of free names Φ of the resulting graph is fully determined by Λ and π (since $\Gamma = \mathrm{dom}(\Lambda)$).

When writing Λ as set of pairs we write the triple $(x, a, \boldsymbol{y})$ for the pair $(x, (a, \boldsymbol{y}))$.

SHR transitions are derived from basic production schemas using suitable sets of inference rules (see §4).

Definition 2.4 (Production schema)
A production schema is an SHR transition of the form:
$x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n) \xrightarrow{\Lambda,\pi} \Phi \vdash G$
where all x_i, $i = 1, \ldots, n$ are distinct.

We suppose to have for each edge label L of arity n a special idle production schema $x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n) \xrightarrow{\Lambda_\epsilon,id} x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n)$ where $\Lambda_\epsilon(x_i) = (\epsilon, \langle\rangle)$ for each i (ϵ is a special "idle" action with $\mathrm{ar}(\epsilon) = 0$). When we have a production schema we suppose to have also all the production

schemas obtainable through α-conversion of names in $\{x_1, \ldots, x_n\} \cup \Phi$.

2.3 Synchronization algebras

We present here synchronization algebras, which were proposed in [15] to deal with synchronizations among processes, as they are performed e.g. in CCS. We use here a slightly different presentation in order to be consistent with standard SHR notation.

In general, we consider a framework where processes can do actions, and actions may or may not synchronize. Suppose that two processes P and P' can do respectively actions a and a'. If actions a and a' synchronize then their synchronized execution corresponds to just one action, otherwise they can not be executed together. Since an action can also be performed asynchronously, we must introduce also an action ϵ that denotes "no action". Thus the synchronized execution of a and ϵ corresponds to the asynchronous execution of a, while P' stays idle.

Definition 2.5 (Synchronization algebra)
A synchronization algebra $< Act, \bullet, \epsilon >$ consists of a binary, partial, associative and commutative operator $\bullet$ on a set of actions Act, which includes a distinguished element ϵ. We require that $\forall a, b \in Act. a \bullet b = \epsilon$ iff $a = b = \epsilon$ and that ϵ is not the only action in Act.

The binary operator $\bullet$ says how actions combine to form synchronized actions: if $a \bullet b$ is undefined then a and b can not synchronize, otherwise $a \bullet b$ is the composed action. The additional condition requires that actions never disappear, thus the result of synchronizing two non ϵ actions can never be ϵ.

3 Synchronization algebras with mobility

We present here a generalization of synchronization algebras [15] which can also deal with mobility and local resource handling. In particular, we are interested in name mobility in the Fusion Calculus [14] style, where the mobility of a process is modelled by merging channels controlled by the process with channels controlled by other processes, thus modelling their proximity.

We attach to each action a tuple of references to channels, and when actions synchronize a pattern of fusions among channels must be specified. References to the resulting channels are attached to the synchronized action. In our framework, a channel may be either public or shared among a group of processes (thus modelling a local resource). On local channels we can have only complete synchronizations, namely synchronizations that do not require any further action. On the other channels we may also have partial synchro-

nizations which will be completed by actions provided by the environment.

We present now the formal definition of synchronization algebra with mobility, which takes into account all these aspects.

As a notation, we use $\uplus$ to denote disjoint set union. In $A \uplus B$ we denote with $[1, n]$ (resp. $[2, n]$) the element that corresponds to $n \in A$ (resp. $n \in B$).

Definition 3.1 (Synchronization algebra with mobility)
A synchronization algebra with mobility $< Act, \bullet, \epsilon, \mathrm{Mob}, Init, Fin >$ *consists of a binary partial operator* $\bullet$ *on a set of actions* Act *which includes a distinguished element* ϵ *of arity* 0*, a set of mobility patterns* Mob *and two subsets* $Init$ *and* Fin *of* Act*. Here* Mob *is a set indexed by pairs of actions, and the element indexed by* (a, b) *is a partial function from* $\{1, \ldots, \mathrm{ar}(a)\} \uplus \{1, \ldots, \mathrm{ar}(b)\}$ *to* $\{1, 2, \ldots\}$.

We require the following conditions to hold:

(i) *the* $\bullet$ *operator is associative and commutative;*

(ii) $\forall a, a' \in Act. a \bullet a' = \epsilon$ *iff* $a = a' = \epsilon$*;*

(iii) $\exists a \in Act. a \neq \epsilon$*;*

(iv) $\epsilon \in Init$*,* $\epsilon \in Fin$*;*

(v) $\forall i \in Init, a \in Act$ *either* $i \bullet a$ *is undefined or* $i \bullet a = a$*;*

(vi) $\forall a \in Act. \exists i \in Init. i \bullet a = a$*;*

(vii) $\forall a, b, c \in Act$
$\forall x \in \{1, \ldots, \mathrm{ar}(a)\}. \mathrm{Mob}_{a \bullet b, c}([1, \mathrm{Mob}_{a,b}([1, x])]) = \mathrm{Mob}_{a, b \bullet c}([1, x])$,
$\forall x \in \{1, \ldots, \mathrm{ar}(b)\}. \mathrm{Mob}_{a \bullet b, c}([1, \mathrm{Mob}_{a,b}([2, x])]) = \mathrm{Mob}_{a, b \bullet c}([2, \mathrm{Mob}_{b,c}([1, x])])$,
$\forall x \in \{1, \ldots, \mathrm{ar}(c)\}. \mathrm{Mob}_{a \bullet b, c}([2, x]) = \mathrm{Mob}_{a, b \bullet c}([2, \mathrm{Mob}_{b,c}([2, x])])$*;*

(viii) $\forall a, b \in Act, x \in \{1, \ldots, \mathrm{ar}(a)\}. \mathrm{Mob}_{a,b}([1, x]) = \mathrm{Mob}_{b,a}([2, x])$*;*

(ix) $\forall a, b \in Act. Mob_{a,b}$ *is surjective on* $\{1, \ldots, \mathrm{ar}(a \bullet b)\}$.

With respect to standard synchronization algebras, now actions in Act have a specified arity, which corresponds to the number of references that are sent with them. A mobility pattern $\mathrm{Mob}_{a,b}$ specifies how to build the references attached to $a \bullet b$ starting from the references attached to a and b. The correspondence is just positional as in usual procedure calls, but many "parameters" can be assigned to just one position. In that case the parameters are merged and a representative is assigned to the chosen position.

Simple message passing is specified by a set of mobility patterns MP that merges corresponding references and assigns the result to the corresponding position. Formally, $MP_{a_1,a_2}([n, x]) = x$ for each $n \in \{1, 2\}$, $x \in \{1, \ldots, \mathrm{ar}(a_n)\}$.

Fin is the set of complete synchronizations. $Init$ is a set of "trivial" actions

which can be done on channels which are not connected to any process. This is required since we want to be able to merge channels, and we want to be always able to synchronize an action i done on an isolated channel with a generic action a without altering a, since no new connection is established when a channel is merged with an isolated channel.

Conditions i, ii, iii are already present in normal synchronization algebras. Condition iv assures that the "no action" is always allowed. Conditions v and vi guarantee respectively that trivial actions never influence synchronization and that given a generic action, there exists a trivial action that can be synchronized with it. Conditions vii and viii state that mobility patterns are associative and commutative. Finally, condition ix guarantees that each reference attached to the composed action can be computed, that is it corresponds to a non empty set of references from component actions.

We present now some simple examples, some more complex ones are in §5. We just write the cases where $\bullet$ is defined. We skip cases that can be derived by commutativity. Furthermore, if not otherwise stated, Mob $= MP$.

Example 3.2 (Milner synchronization algebra)
Given a set of actions $Act = \{a_i\}_{i\in I} \cup \{\overline{a_i}\}_{i\in I} \cup \{\tau, \epsilon\}$ *where* $\mathrm{ar}(\overline{a_i}) = \mathrm{ar}(a_i)$ *and* $\mathrm{ar}(\tau) = 0$ *we define the Milner synchronization algebra on Act as follows:*

- $a \bullet \epsilon = a$ *for each* $a \in Act$,
  $a \bullet \overline{a} = \tau$ *for each* $a \in \{a_i\}_{i\in I}$;
- $Fin = \{\tau, \epsilon\}$, $Init = \{\epsilon\}$.

The Milner synchronization algebra models message passing, where actions $a_i/\overline{a_i}$ are inputs/outputs and τ stands for a complete message exchange.

Example 3.3 (Hoare synchronization algebra)
Given a set of actions Act we define the Hoare synchronization algebra on Act as follows:

- $a \bullet a = a$ *for each* $a \in Act$;
- $Fin = Act$, $Init = Act$.

The Hoare synchronization algebra models an agreement among participants on the action to be done, in the CSP style.

Example 3.4 (Broadcast synchronization algebra)
Given a set of actions $Act = \{a_i\}_{i\in I} \cup \{\overline{a_i}\}_{i\in I} \cup \{\epsilon\}$ *where* $\mathrm{ar}(\overline{a_i}) = \mathrm{ar}(a_i)$ *we define the broadcast synchronization algebra on Act as follows:*

- $a \bullet \overline{a} = \overline{a}$ *for each* $a \in \{a_i\}_{i\in I}$,
  $a \bullet a = a$ *for each* $a \in \{a_i\}_{i\in I}$,
  $\epsilon \bullet \epsilon = \epsilon$;

- $Fin = \{\overline{a_i}\}_{i\in I} \cup \{\epsilon\}$, $Init = \{a_i\}_{i\in I} \cup \{\epsilon\}$.

The broadcast synchronization algebra models secure broadcast, where one process performs an output and all the others perform input.

4 Parametric Synchronized Hyperedge Replacement

We present here a set of rules for deriving SHR transitions from productions, which is parametric on a synchronization algebra with mobility. Thus we can instantiate the rules for each synchronization model.

As already said, in our approach edges model processes, nodes model channels and free nodes model public channels. It is worth noting that nodes are bound when they are created, since they are not known by the environment. A bound node becomes free (we say that it is extruded) when it is merged with a free node or when a reference to it is exposed on a free node.

As notation we use $f|_S$ (resp. $f|_{\setminus S}$) to denote the restriction of function f to the new domain S (resp. $\mathrm{dom}(f) \setminus S$). We also use $\boldsymbol{v}[n]$ to denote the nth element of vector $\boldsymbol{v}$ and $\mathrm{Set}(\boldsymbol{v})$ to denote the set of elements of $\boldsymbol{v}$.

Definition 4.1 (Rules for parametric SHR)

$$\textit{(par)}\quad \frac{\Gamma_1 \vdash G_1 \xrightarrow{\Lambda_1,\pi_1} \Phi_1 \vdash G_1' \qquad \Gamma_2 \vdash G_2 \xrightarrow{\Lambda_2,\pi_2} \Phi_2 \vdash G_2'}{\Gamma_1,\Gamma_2 \vdash G_1|G_2 \xrightarrow{\Lambda_1\cup\Lambda_2,\pi_1\cup\pi_2} \Phi_1,\Phi_2 \vdash G_1'|G_2'}$$

if $(\Gamma_1 \cup \mathrm{n}(\Lambda_1)) \cap (\Gamma_2 \cup \mathrm{n}(\Lambda_2)) = \emptyset$.

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

where:

- $\sigma = \{x/y\}$;
- $\Lambda(x) = (a_1, \boldsymbol{v}_1), \Lambda(y) = (a_2, \boldsymbol{v}_2)$;
- $c = a_1 \bullet a_2$;
- $S_1 = \{\boldsymbol{v}_{i_1}[j_1] = \boldsymbol{v}_{i_2}[j_2] | \mathrm{Mob}_{a_1,a_2}([i_1, j_1]) = \mathrm{Mob}_{a_1,a_2}([i_2, j_2])\}$;
- $S_2 = \{t = u | t, u \in \Gamma, x, y \wedge t\pi = u\pi\}$;
- $\rho = \mathrm{mgu}(\{(S_1 \cup S_2)\sigma\})$ *where we choose nodes in* Γ, x *as representatives whenever possible;*
- $\boldsymbol{w}[i] = (\boldsymbol{v}_j[k])\sigma\rho$ *if* $\mathrm{Mob}_{a_1,a_2}([j, k]) = i$, $i \leq \mathrm{ar}(c)$;

- $\Lambda'(z) = \begin{cases} (c, \boldsymbol{w}) \text{ if } z = x \\ (\mathrm{act}_\Lambda(z), (\mathrm{n}_\Lambda(z))\sigma\rho) \text{ for each } z \in \Gamma \end{cases}$
- $\pi' = \rho|_{\Gamma,x}$;
- $U = \Phi\sigma\rho \setminus \Phi'$.

$$\textit{(res)} \quad \frac{\Gamma, x \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'}{\Gamma \vdash \nu x\ G \xrightarrow{\Lambda|_{\setminus\{x\}},\pi|_{\setminus\{x\}}} \Phi' \vdash \nu Z\ G'}$$

where:

- $(x\pi = y\pi \wedge x \neq y) \Rightarrow x\pi \neq x$;
- $\mathrm{act}_\Lambda(x) \in Fin$;
- $Z = \Phi \setminus \Phi'$.

$$\textit{(new)} \quad \frac{\Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'}{\Gamma, x \vdash G \xrightarrow{\Lambda \cup \{(x,i,\boldsymbol{y})\},\pi} \Gamma, x, \mathrm{Set}(\boldsymbol{y}) \vdash G'}$$

for each $i \in Init$ *and each vector* $\boldsymbol{y}$ *of names such that* $\mathrm{Set}(\boldsymbol{y}) \cap (\Gamma, x \cup \mathrm{n}(\Lambda)) = \emptyset$ *and* $\mathrm{ar}(i) = |\boldsymbol{y}|$.

Rule (par) allows to perform the union of two transitions provided that they have disjoint sets of free names (accounting also for newly generated names).

Rule (merge) is the rule for synchronization. It allows to compute the effect of merging two nodes x and y with synchronizations $(a_1, \boldsymbol{v}_1)$ and $(a_2, \boldsymbol{v}_2)$ respectively on them. The synchronization is allowed iff the composed action $a_1 \bullet a_2$ is defined. In this case two sets of equations among names are computed. S_1 accounts for merging names that are mapped to the same position by Mob (note that merges are performed even if the resulting representative is not attached to the final action), while S_2 accounts for previous merges traced by π. We then compute ρ by applying the substitution σ to S_1 and S_2 and then choosing a representative for each equivalence class. If at least one of the members of the class is in Γ, x, then one of them must be chosen (otherwise unjustified renamings of nodes may happen). After that the new vector $\boldsymbol{w}$ is generated, by choosing for each position the representative of the corresponding equivalence class. We can then compute the new synchronization Λ', which takes into account the performed merges. Merges on nodes in the interface are traced by π'. Finally, nodes which are no more extruded (because the synchronization discarded them) are bound.

Rule (res) binds nodes. According to the first condition, the bound node must not be the representative of the equivalence class induced by π when

the class is not trivial. Furthermore a node can be bound only if a complete action takes place on it. Nodes that were extruded just on the bound node must be bound now, and thus they are in Z.

Rule (new) allows to add an isolated node to the interface, on which the trivial actions in $Init$ can be exposed, with fresh names.

We show here that this framework can be instantiated in order to recover the SHR models in the literature.

Theorem 4.2 *Let $\Gamma \vdash G$ be a graph. With a fixed set of productions, we can derive a transition $\Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$ using the inference rules in Definition 4.1 (excluded (new)) instantiated with the Milner synchronization model iff we can derive, using the rules in [4], a transition $\Gamma \vdash G \xrightarrow{\Lambda',\pi} \Phi \vdash G'$ where $\Lambda'(x) = \Lambda(x)$ iff $\Lambda(x) \neq (\epsilon, \langle\rangle)$ and $\Lambda'(x)$ is undefined otherwise.*

The theorem states that the framework presented in [4] is an instance of ours with Milner synchronization, where undefined is used instead of $(\epsilon, \langle\rangle)$.

Theorem 4.3 *Let $\Gamma \vdash G$ and G' be graphs without restrictions. With a set of productions S, we can derive a transition $\Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi' \vdash \nu X\ G'$ using the inference rules in Definition 4.1 instantiated with the Hoare synchronization model iff we can derive a transition $\Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi', X \vdash G'$ using the rules in [12] (or [10]) and a set of productions obtained from S by removing all the restrictions (after having chosen fresh names for bound names).*

This theorem states that the framework in [12,10] is an instance of this one with Hoare synchronization, where restriction is not considered.

5 Examples

We show here some examples on how to use SHR and on how to define synchronization algebras with mobility. We start with a simple example that uses the Hoare synchronization model.

Example 5.1 (The replicating net) *We define here a simple net that can evolve when the same action is exposed on all the nodes in its interface. If the used action has arity n then the net creates n copies of itself, attached to the nodes whose references were exposed on the interface.*

We must have one production schema for each allowed action and each edge label. We show here just the production for an action a with $\mathrm{ar}(a) = 1$

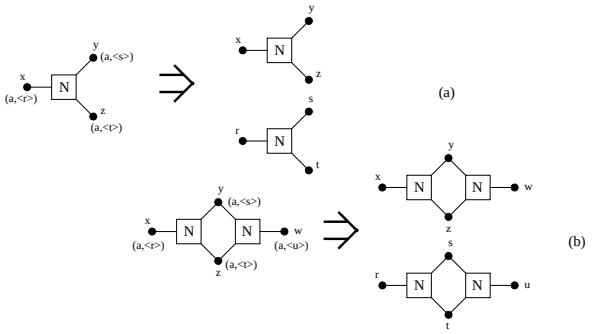

Fig. 1. Production (a) and transition (b) of a replicating net.

and an edge labelled by N *with* $\mathrm{rank}(N) = 3$.

$$x, y, z \vdash N(x, y, z) \xrightarrow{\{(x,a,\langle r\rangle),(y,a,\langle s\rangle),(z,a,\langle t\rangle)\},id} x, y, z, r, s, t \vdash N(x, y, z)|N(r, s, t)$$

A graphical presentation of the production is in Figure 1(a), while in Figure 1(b) we have a simple transition.

In order to derive this transition we must take two suitably renamed instances of the production schema, and then use rule (par) obtaining:

$$x, y, z, w, y_1, z_1 \vdash N(x, y, z)|N(w, y_1, z_1) \xrightarrow{\{(x,a,\langle r\rangle),(y,a,\langle s\rangle),(z,a,\langle t\rangle)(w,a,\langle u\rangle),(y_1,a,\langle s_1\rangle),(z_1,a,\langle t_1\rangle)\},id} x, y, z, r, s, t, w, y_1, z_1, u, s_1, t_1 \vdash N(x, y, z)|N(r, s, t)|N(w, y_1, z_1)|N(u, s_1, t_1)$$

We can then apply two times rule (merge) with $\sigma = \{y/y_1\}$ *and* $\sigma = \{z/z_1\}$ *obtaining at the end:*

$$x, y, z, w \vdash N(x, y, z)|N(w, y, z) \xrightarrow{\{(x,a,\langle r\rangle),(y,a,\langle s\rangle),(z,a,\langle t\rangle)(w,a,\langle u\rangle)\},id} x, y, z, r, s, t, w, u \vdash N(x, y, z)|N(r, s, t)|N(w, y, z)|N(u, s, t)$$

as desired. Note that when the first (merge) rule is applied we have $S_1 = \{s = s_1\}$ *thus we can choose* $\rho = \{s/s_1\}$.

A realistic scenario that can be modelled by such a transformation is for instance a server process that accepts communications on a port, which can be modelled as a graph with just one free node, and that, when required from the client, spawns a copy of itself to manage further requests using another port.

Example 5.2 (The Game of Life) [3] *The Game of Life [1] is a well-known cellular automaton, that is a grid of evolving cells. A cell can be empty or populated (alive). A living cell dies if it has one or no alive neighbours (loneliness) or if it has four or more alive neighbours (overpopulation) and survives otherwise. An empty cell becomes alive if it has exactly three alive neighbours.*

In order to model the Game of Life we use edges that represent cells, with labels A for alive and E for empty. Edges are connected to their eight neighbours via shared nodes.

At each step edges must communicate their state on each link and, at the same time, receive the state of their neighbours. This can be done using four actions (e,e),(a,a),(e,a) *and* (a,e). *The meaning of e.g. the last action is "I'm alive, you are empty". We also introduce an action ok representing a successful synchronization. Since the network is static we have no mobility, i.e. all actions have arity* 0. *We use the following synchronization algebra (with mobility):*

- $Act = \{(e,e),(a,a),(e,a),(a,e),ok,\epsilon\}$;
- $\epsilon \bullet a = a$ *for each* $a \in Act$,
  $(e,e) \bullet (e,e) = ok$, $(e,a) \bullet (a,e) = ok$, $(a,a) \bullet (a,a) = ok$;
- $Fin = \{\epsilon, ok\}$, $Init = \{\epsilon\}$;
- Mob *contains just functions from* $\emptyset \uplus \emptyset$ *to* $\emptyset$.

In order to force complete synchronizations all internal nodes must be bound. We use for productions a simplified notation: we just write the labels of the edges and the sequence of actions. Here we have some productions:

loneliness $A \xrightarrow{(a,e),(a,a),(a,e),(a,e),(a,e),(a,e),(a,e),(a,e),(a,e)} E$

survive $A \xrightarrow{(a,e),(a,a),(a,e),(a,a),(a,e),(a,a),(a,e),(a,e),(a,e)} A$

overpopulation $A \xrightarrow{(a,a),(a,a),(a,e),(a,a),(a,e),(a,a),(a,a),(a,a),(a,e)} E$

populate $E \xrightarrow{(e,a),(e,e),(e,e),(e,a),(e,e),(e,e),(e,e),(e,a),(e,e)} A$

Example 5.3 (Threshold synchronization) *We define here an algebra for threshold synchronization that allows a group of processes to get some information from a sender, but only if at least* m *of them agree.*

The synchronization algebra with mobility is defined as follows:

- $Act = \{out, \epsilon\} \cup \{(in,n) | n \in \mathbb{N}\} \cup \{(in*,n) | n \in \mathbb{N}\}$;
- out *and* (in,n) *have arity* 1, $(in*,n)$ *has arity* 2 *for* $n < m$, 1 *otherwise;*
- $\epsilon \bullet a = a \ \forall a \in Act$,

[3] We thanks Robin Milner, who suggested the example during Dagstuhl seminar 04241.

$out \bullet (in, n) = (in*, n)$,
$(in, n_1) \bullet (in, n_2) = (in, n_1 + n_2)$,
$(in, n_1) \bullet (in*, n_2) = (in*, n_1 + n_2)$;

- $Fin = \{\epsilon\} \cup \{(in*, n) | n \in \mathbb{N}\}$, $Init = \{\epsilon\}$;
- Mob *always maps the parameter of* (in, n) *and the first parameter of* $(in*, n)$ *to the first parameter of the result, and the parameter of out and the second parameter of* $(in*, n)$ *(whenever defined) to the second parameter of the resulting* $(in*, n)$ *if* $n < m$*, to the only one otherwise.*

Suppose that processes can perform only the actions out and $(in, 1)$*, the others being auxiliary. During synchronization the inputs are merged, but until* m *of them have joined, the parameter of the output is kept separate. Thus in order for the processes doing input to get the piece of information in the output, at least* m *of them must participate to the synchronization. If they are not enough synchronization is performed but no data exchange occurs.*

6 A case study on Fusion Calculus

In this section we briefly summarize Fusion Calculus, an important calculus for mobility that extends π-calculus with the concept of fusion, and we show an encoding from (a subset of) Fusion Calculus processes to graphs. An important aspect of our encoding is that the topology of the graph does not correspond to the structure of the syntactic tree of the process, but it models the structure of the connections among processes. We show that, using synchronization algebras, we can provide different concurrent semantics for Fusion Calculus, just by choosing different synchronization models.

For a full definition of Fusion Calculus see [14], and for a more detailed analysis of the relationships between Fusion Calculus and SHR see [11].

We consider agents with the following syntax.

Definition 6.1 *The agents are defined by:*
$S ::= \sum_i \alpha_i.P_i \mid \mathrm{rec}\, X.\alpha.P \mid X$ *(Sequential agents)*
$P ::= 0 \mid S \mid P_1|P_2 \mid (x)P$ *(General agents)*
where α *are input/output actions* $u\boldsymbol{x}$ / $\overline{u}\boldsymbol{x}$ *or fusion actions* ϕ *(equivalence relations on names).*

Processes are agents considered up to standard structural axioms. For the definition of the operational semantics see [14]. We just remember that fusion transitions are of the form $P \xrightarrow{\alpha} P'$ where α is one of the actions seen above or a bound communication action, that is an input/output action where names in a set $\mathrm{bn}(\alpha)$ are extruded.

Our mapping uses the following standard decomposition on processes.

Definition 6.2 *The standard decomposition of a process P is defined as $P = \hat{P}\sigma_P$ where σ_P is the standard substitution and $\hat{P}$ is the standard agent of P. No free name can occur twice in $\hat{P}$. The decomposition satisfies $P = Q\sigma \Rightarrow \hat{P} = \hat{Q} \wedge \sigma_P = \sigma_Q\sigma$. We denote with* $\mathrm{fnarray}(P)$ *the array of the free name occurrences in P. In particular $\sigma_P = \{\mathrm{fnarray}(P)/\,\mathrm{fnarray}(\hat{P})\}$.*

We present now the relationships between Fusion Calculus and SHR.

Definition 6.3 *A substitutive effect of a fusion ϕ is an idempotent substitution σ agreeing with ϕ (i.e. σ sends all members of each equivalence class of ϕ to one representative in the class).*

Definition 6.4 *The relation between actions of Fusion Calculus and SHR synchronizations is a function that we denote with $[\![-]\!]$ in the case of communication actions:*
$[\![(\boldsymbol{y})u\boldsymbol{x}]\!] = (u, in_n, \boldsymbol{x}), id$ *where* $n = |\boldsymbol{x}|$
$[\![(\boldsymbol{y})\overline{u}\boldsymbol{x}]\!] = (u, out_n, \boldsymbol{x}), id$ *where* $n = |\boldsymbol{x}|$
When not otherwise stated, we suppose to have on nodes the trivial synchronization $(\epsilon, \langle\rangle)$. We define in_n and out_n as complementary actions, i.e. $in_n = \overline{out_n}$ and $out_n = \overline{in_n}$. A ϕ action corresponds to a substitutive effect π of ϕ.

Definition 6.5 *We define now the translation on processes:*

$$[\![0]\!] = nil \quad [\![S]\!] = L_{\hat{S}}(\mathrm{fnarray}(S)) \quad [\![P_1|P_2]\!] = [\![P_1]\!]|[\![P_2]\!] \quad [\![(x)P]\!] = \nu x\ [\![P]\!]$$

As a last step we show the production schemas used for the SHR system.

Definition 6.6 *We have production schemas only for standard sequential agents $\sum_i \alpha_i.P_i$. Let Γ be $\mathrm{fn}([\![\sum_i \alpha_i.P_i]\!])$. Productions are of the following forms:*
$\Gamma \vdash [\![\sum_i \alpha_i.P_i]\!] \xrightarrow{[\![\alpha_i]\!]} \Gamma \vdash [\![P_i]\!]$ *if α_i is a communication action;*
$\Gamma \vdash [\![\sum_i \alpha_i.P_i]\!] \xrightarrow{\pi} \Gamma\pi \vdash [\![P_i\pi]\!]$ *if $\alpha_i = \phi$ and π is a substitutive effect of ϕ.*

The correctness of the translation w.r.t. SHR with Milner synchronization is proved by the following theorem.

Theorem 6.7 (Correctness) *Let us consider SHR with Milner synchronization algebra. For each fusion process P and each $\Gamma \supseteq \mathrm{fn}([\![P]\!])$ if $P \xrightarrow{\alpha} P'$ and $\Gamma \cap \mathrm{bn}(\alpha) = \emptyset$ then:*

(i) *if α is a communication action then $\Gamma \vdash [\![P]\!] \xrightarrow{[\![\alpha]\!]} \Gamma, \Gamma_E \vdash [\![P']\!]$ where $\Gamma_E = \mathrm{bn}(\alpha)$;*

(ii) *if α is a fusion action then $\Gamma \vdash [\![P]\!] \xrightarrow{\pi} \Gamma\pi \vdash [\![P'\pi]\!]$ for each substitutive effect π of α. We may or may not have also a τ action on the node in*

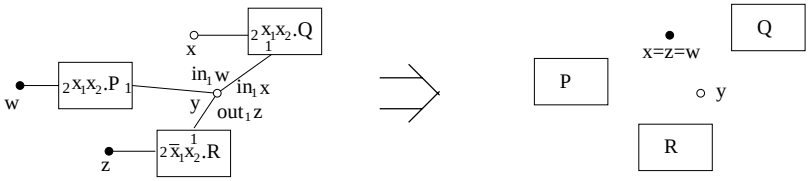

Fig. 2. A transition of Broadcast Fusion.

Γ *on which the synchronization has been performed (if the node is bound then the* τ *action is not visible).*

In general our graphs can perform more transitions than the corresponding Fusion Calculus processes. This happens because SHR is a concurrent model and thus many Fusion Calculus transitions can be executed in one SHR step.

Example 6.8 *Let us consider the fusion process* $(xy)(yw.P|yx.Q|\overline{y}z.R)$. *We can have for instance the following transition:*

$$(xy)(yw.P|yx.Q|\overline{y}z.R) \xrightarrow{1} (y)((yw.P|Q|R)\{z/x\})$$

Using the translation and the Milner synchronization algebra we can have in the SHR model the corresponding transition:

$$w, z \vdash \nu x, y\ L_{x_1x_2.P}(y, w)|L_{x_1x_2.Q}(y, x)|L_{\overline{x_1}x_2.R}(y, z) \xrightarrow{id} w, z \vdash \nu y\ (L_{x_1x_2.P}(y, w)|L_Q|L_R)\{z/x\}$$

One can use the same approach to analyze systems based on different communication primitives. Suppose that we have a system based on broadcast. We can derive a Broadcast Fusion Calculus by applying the translation and animating the obtained graph using SHR with broadcast synchronization. Thus we can have for instance a transition of the form:

$$w, z \vdash \nu x, y\ L_{x_1x_2.P}(y, w)|L_{x_1x_2.Q}(y, x)|L_{\overline{x_1}x_2.R}(y, z) \xrightarrow{\{z/w\}} z \vdash \nu y\ (L_P|L_Q|L_R)\{z/x, z/w\}$$

The same transition is graphically represented in Figure 2 and can be written in the usual Fusion Calculus notation as:

$$(xy)(yw.P|yx.Q|\overline{y}z.R) \xrightarrow{\{z/w\}} (y)(P|Q|R)\{z/x\}$$

7 Conclusions and future work

We have presented a general model for SHR, where the synchronization mechanism and the mobility patterns are specified using a generalization of syn-

chronization algebras. We have presented some examples in order to prove that synchronization algebras with mobility are a suitable way to model the communication primitives provided by the middleware, and that SHR can be easily used to build models of systems at the high level of abstraction, as required for the huge systems that are met in the global computing area.

We have also presented a mapping from Fusion Calculus to SHR, showing that SHR is a good framework to execute fusion processes, since one can easily change the synchronization model, thus allowing to use the usual Fusion notation while modelling systems based on different communication primitives.

As future work we want to define an abstract semantics for SHR, thus allowing a behavioural analysis of graph computations and allowing to prove that different graphs are equivalent w.r.t. some concept of observation.

We plan to define a framework where each node is labelled with a synchronization algebra, thus allowing to model heterogeneous systems that use different communication primitives inside the same system, as often happens in the global computing scenario.

We also want to analyze expressiveness and properties of the different kinds of Fusion Calculi obtained through our mapping, first of all the broadcast one.

Finally, we want to build an implementation of SHR which allows to draw graphs and productions and to animate them in order to analyze their possible behaviours.

References

[1] E. Berlekamp, J. Conway, and R. Guy. *Winning Ways for your Mathematical Plays*, volume 2. Academic Press, 1982.

[2] P. Degano and U. Montanari. A model for distributed systems based on graph rewriting. *Journal of the ACM (JACM)*, 34(2):411–449, 1987.

[3] H. Ehrig, M. Pfender, and H. J. Schneider. Graph grammars: an algebraic approach. In *Proc. IEEE Conference on Automata and Switching Theory*, pages 167–180, 1973.

[4] G. Ferrari, U. Montanari, and E. Tuosto. A LTS semantics of ambients via graph synchronization with mobility. In *Proc. of ICTCS'01*, volume 2202 of *LNCS*, pages 1–16. Springer, October 2001.

[5] O. H. Jensen and R. Milner. Bigraphs and transitions. *SIGPLAN Not.*, 38(1):38–49, 2003.

[6] D. Hirsch. *Graph Transformation Models for Software Architecture Styles*. PhD thesis, Departamento de Computación, U.B.A., 2003.

[7] D. Hirsch, P. Inverardi, and U. Montanari. Reconfiguration of software architecture styles with name mobility. In *Proc. of Coordination 2000*, volume 1906 of *LNCS*, February 2000.

[8] D. Hirsch and U. Montanari. Synchronized hyperedge replacement with name mobility. In *Proc. of CONCUR'01*, Springer, volume 2154 of *LNCS*, 2001.

[9] I. Lanese. Process synchronization in distributed systems via Horn clauses. Master's thesis, University of Pisa, Computer Science Department, 2002. Downloadable from `http://www.di.unipi.it/~lanese/work/tesi.ps`.

[10] I. Lanese and U. Montanari. Software architectures, global computing and graph transformation via logic programming. In *Proc SBES'2002*, pages 11–35. Anais, 2002.

[11] I. Lanese and U. Montanari. A graphical fusion calculus. In *Proceedings of CoMeta Final Workshop*, pages 199–215, volume 104 of ENTCS, 2004.

[12] I. Lanese and U. Montanari. Mapping fusion and synchronized hyperedge replacement into logic programming. *Theory and Practice of Logic Programming, Special Issue on Multiparadigm Languages and Constraint Programming*, 2004. Submitted.

[13] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Inform. and Comput.*, 100:1–77, 1992.

[14] J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proc. of LICS '98*. IEEE, June 1998.

[15] G. Winskel. Event structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, volume 255 of *LNCS*, pages 325–392. Springer, 1986.