Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 211 (2008) 27–38
www.elsevier.com/locate/entcs

Exploiting User-definable Synchronizations in Graph Transformation*

Ivan Lanese[1]

Computer Science Department, University of Bologna, Bologna, Italy

Abstract

Parametric Synchronized Hyperedge Replacement (PSHR) is a *graph transformation* formalism where productions specifying the behavior of single components can be synchronized to give full transitions. The main feature of PSHR is that *the synchronization model is user-definable*. To enhance the applicability of the approach we propose a simplified and more suggestive semantics, preserving however the expressive power of the original one. We also show how some common synchronization models can be formalized and exploited inside PSHR. This allows to simplify the modelling step, and the produced model too. We apply this approach to the airport case study of FET-GC project AGILE.

Keywords: Graph transformation, Synchronized Hyperedge Replacement, synchronization algebras, mobility.

1 Introduction

Architectural modelling is the step of the design of a system that fixes the structure of the system, that is its components and the connections among them, and its evolution over time. Since these aspects have a large impact on all the following phases of the development process, it is important that the decisions made are clearly stated in the model. This requires modelling frameworks with a formal syntax and semantics.

Many approaches to this problem have been presented in the literature, from UML [13] to different Architecture Description Languages [3]. We choose as framework *Synchronized Hyperedge Replacement* (SHR) [6], which is a *graph transformation* framework. Thus the system is modelled as an (hyper)graph, where (hyper)edges are components connected through common nodes. This provides both sound mathematical foundations and a suggestive visual representation. In SHR the behavior of components is specified by productions which can be synchronized

* Research supported by the EC Project IST-FP6 16004 SENSORIA.

[1] Email: lanese@cs.unibo.it

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2008.04.027

to build transitions. In particular, productions perform actions on nearby nodes, and actions performed on the same node must be compatible according to some *synchronization model*. We also use *mobility* [9,8], allowing actions to carry nodes as parameters, and synchronization to merge them thus reconfiguring the system. In particular, we consider Parametric SHR (PSHR) [11], where both action synchronization and mobility patterns are specified by a user-defined *Synchronization Algebra with Mobility* (SAM). This allows to choose each time the most suitable synchronization model for the application at hand.

PSHR is very expressive, as shown in [11], but its formal semantics is quite heavy and difficult to understand. This problem is common to other SHR variants, and is aggravated in PSHR by the need to manage different synchronization models. The problem is due to the fact that the standard semantics of SHR is based on inference rules that exploit a representation of graphs as terms in an algebra. In this presentation each part of the transition is obtained as a result of many inference steps, thus it is not easy to guess the global effect of a set of productions. We propose a more extensional semantics, where the synchronizations allowed on a node by a specific SAM are directly characterized, and an algorithm specifies how to build a full transition. Also, the semantics is based on a set-theoretic representation of graphs instead of on an algebraic one.

We also show how a synchronization model can be formalized as a SAM, and how PSHR can be used to model the evolution of a system using this SAM. We apply this approach to the airport case study [2], which has been proposed inside the FET-GC project AGILE [1] on architectures for mobility. We show that parametric synchronization allows a simpler model than the one presented in [4], where a synchronized version of Double Pushout [7] based on a fixed two-parties synchronization is used.

Structure of the paper. § 2 defines graphs and SHR transitions. § 3 presents SAMs, characterizes their effects, and analyzes the modelling of synchronization policies as SAMs. § 4 contains the algorithm to derive transitions from productions. § 5 details the application of the approach to the airport case study. Finally, § 6 presents conclusions and plans for future work.

2 Hypergraphs and SHR transitions

SHR [6] is an approach to (hyper)graph transformation that defines *global transitions* using *local productions*. Productions define how a single (hyper)edge can be rewritten and the *conditions* that this rewriting imposes. Conditions are specified as compatibility requirements among *actions* performed by productions on nearby nodes. The exact requirements depend on the chosen synchronization model. We use the extension of SHR with *mobility* [9,8], that allows edges to send node references together with actions, and nodes whose references are matched during synchronization are merged. In this work we use Parametric SHR (PSHR) [11], where the used synchronization model and mobility patterns can be freely chosen by specifying them via a Synchronization Algebra with Mobility (SAM). A detailed

description of different SHR frameworks can be found in [10].

The usual presentation of SHR is based on a representation of graphs as terms in a suitable term algebra and on inference rules to derive transitions from productions. This presentation allows to easily prove properties of the framework exploiting techniques from the process calculi field, but it is not so suggestive, since transitions are built as a result of many inference steps, and this makes difficult to understand the actual interactions. Also, the general mechanism is hidden because of heavy technicalities.

We propose here an original and more suggestive semantics, where transitions are built using an ad-hoc algorithm that highlights the main features of the synchronization and mobility mechanisms, and we present a direct description of the interactions allowed by a SAM. Also, our semantics is based on a set-theoretic presentation of graphs instead of on an algebraic one.

We always assume to have a countable set of nodes $\mathcal{N}$, a countable set of edges $\mathcal{E}$, and a countable ranked set of edge labels LE. Given $L \in LE$, $\mathrm{rank}(L)$ is its rank.

Definition 2.1 (Hypergraph)
A (hyper)graph *is a tuple* $\langle E, \mathrm{lab}, N, \mathrm{conn}, \Gamma\rangle$ *where* $E \subseteq \mathcal{E}$ *is the set of edges,* $\mathrm{lab} : E \to LE$ *is the labelling function for edges,* $N \subseteq \mathcal{N}$ *is the set of nodes,* $conn : E \to N^*$ *is a function mapping each edge* e *to a* n*-uple of nodes where* n *is the rank of the label* $\mathrm{lab}(e)$*, and* $\Gamma \subseteq N$ *is the set of nodes in the interface. Nodes not in the interface are said hidden.*

Graphs are considered up to bijective renamings of edges and of hidden nodes.

In the above description *conn* specifies to which nodes each edge is attached. We present now the steps of an SHR computation.

Definition 2.2 (SHR transition) *Let Act be a set of actions, and given* $a \in Act$ *let* $\mathrm{ar}(a)$ *be its arity. An* SHR transition *is of the form:*

$$G \xrightarrow{\Lambda,\pi} G'$$

where G *and* G' *are graphs. Let* Γ_G *be the interface of* G*. Then* $\Lambda : \Gamma_G \to (Act \times \mathcal{N}^*)$ *is a total function and* $\pi : \Gamma_G \to \Gamma_G$ *is an idempotent substitution. Function* Λ *assigns to each node* x *the action* $a \in Act$ *and the vector* $\boldsymbol{y}$ *of node references sent to* x *by the transition. If* $\Lambda(x) = \langle a, \boldsymbol{y}\rangle$ *then we define* $\mathrm{n}_\Lambda(x) = \boldsymbol{y}$*. We require that* $\mathrm{ar}(a) = |\boldsymbol{y}|$*. We define the set of communicated names* $\mathrm{n}(\Lambda)$ *as* $\{z | \exists x.z \in \mathrm{n}_\Lambda(x)\}$*. 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* $\forall x \in \mathrm{n}(\Lambda).x\pi = x$*, i.e., only references to representatives can be sent.*

SHR transitions are obtained by synchronizing productions using a specified synchronization model.

Definition 2.3 (SHR production) *An* SHR production *is an SHR transition* $G \xrightarrow{\Lambda,\mathrm{id}} G'$ *such that* G *is a graph containing exactly one edge* e*. Also, each node in*

G occurs exactly once in $\mathrm{conn}(e)$*. Furthermore the node substitution in the label is* id *and the interface of* G' *is* $\Gamma_G \cup \mathrm{n}(\Lambda)$*.*

For each G of the above form there is an idle production $G \xrightarrow{\Lambda_\epsilon,\mathrm{id}} G$ *where* $\Lambda_\epsilon(x) = \langle \epsilon, \langle\rangle\rangle$ *for each* $x \in \Gamma_G$ *(ϵ is a special "idle" action with* $\mathrm{ar}(\epsilon) = 0$*). Idle productions are included in all sets of productions, which are also closed under bijective renamings of nodes.*

3 Synchronization Algebras with Mobility

We formalize a synchronization model as a Synchronization Algebra with Mobility (SAM). SAMs were first introduced in [11], extending Winskel's synchronization algebras (SAs) [14] to deal with mobility of nodes.

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

Definition 3.1 (Synchronization algebra with mobility)
A Synchronization Algebra with Mobility $< Act, \mathrm{ar}, \bullet, \epsilon, \mathrm{mob}, Fin >$ *consists of a binary partial operator $\bullet$ on a set of actions Act, a set of mobility patterns* mob *and a subset Fin of Act. Function* $\mathrm{ar} : Act \to \mathbb{N}$ *maps each action $a \in Act$ to its arity* $\mathrm{ar}(a)$*, and $\epsilon \in Act$ is an action of arity* 0*. Here* mob *is a set indexed by pairs of actions (a, b) such that $a \bullet b$ is defined, and* $\mathrm{mob}_{a,b}$ *is a partial function from* $\{1, \ldots, \mathrm{ar}(a)\} \uplus \{1, \ldots, \mathrm{ar}(b)\}$ *to* $\mathbb{N}$*.*

We impose the following conditions:

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

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

(iii) $\forall a \in Act. a \bullet \epsilon$ *is defined* $\Rightarrow$
$(a \bullet \epsilon = a \wedge \forall x \in \{1, \ldots, \mathrm{ar}(a)\}. \mathrm{mob}_{a,\epsilon}([1, x]) = x)$*;*

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

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

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

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

As in SAs, we have a set of actions Act and an operator $\bullet$ of action composition. Here $a \bullet b = c$ means that actions a and b can synchronize giving action c as a result. If $a \bullet b$ is undefined then a and b are not compatible. For instance in CCS an action a can synchronize with a coaction $\overline{a}$ producing τ as a result. Action ϵ stands for "not taking part to the synchronization", and it allows to specify in a uniform way action synchronization and asynchronous execution of actions. In fact, $a \bullet \epsilon = a$ means that a is executed asynchronously. With respect to SAs, now actions a in Act have a specified arity $\mathrm{ar}(a)$, which corresponds to the number of node references carried

by a. 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 the result is assigned to the chosen position. Using $\mathbb{N}$ as codomain instead of $\{1, \ldots, \mathrm{ar}(a \bullet b)\}$ allows to specify merges among parameters even if the chosen representative of the equivalence class defined in this way does not occur in the final label.

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 $n \in \{1,2\}$, $x \in \{1, \ldots, \mathrm{ar}(a_n)\}$.

A mobility pattern mob_{a_1,a_2} included in a SAM S can be applied to two actions $\langle a_1, \boldsymbol{y}_1 \rangle$ and $\langle a_2, \boldsymbol{y}_2 \rangle$ to compute both the substitution σ performing the merge of parameters and the vector of parameters of the result, given respectively by the two functions:

$$\sigma = \mathrm{sub}(S, \langle a_1, \boldsymbol{y}_1 \rangle, \langle a_2, \boldsymbol{y}_2 \rangle) = \mathrm{mgu}(\{\boldsymbol{y}_i[j] = \boldsymbol{y}_h[k] \mid \mathrm{mob}_{a_1,a_2}([i,j]) = \mathrm{mob}_{a_1,a_2}([h,k])\})$$

$$\mathrm{par}(S, \langle a_1, \boldsymbol{y}_1 \rangle, \langle a_2, \boldsymbol{y}_2 \rangle)[i] = (y_h[k])\sigma \quad \text{where } h,k \text{ are such that } \mathrm{mob}_{a_1,a_2}([h,k]) = i \wedge i \leq \mathrm{ar}(a_1 \bullet a_2)$$

For instance, let S be a message-passing SAM with actions a, b and c of arity 1, 3 and 2 respectively, such that $a \bullet b = c$. Then $\mathrm{sub}(S, \langle a, \langle x_1 \rangle \rangle, \langle b, \langle y_1, y_2, y_3 \rangle \rangle) = \{x_1/y_1\}$ (also $\{y_1/x_1\}$ is a valid choice) and $\mathrm{par}(S, \langle a, \langle x_1 \rangle \rangle, \langle b, \langle y_1, y_2, y_3 \rangle \rangle) = \langle x_1, y_2 \rangle$. If we consider an action a' of arity 3 with parameters $\langle x_1, x_2, x_3 \rangle$ instead of a, then $\sigma = \{x_1/y_1, x_2/y_2, x_3/y_3\}$, but x_3 is not a parameter of the resulting action.

Fin is the set of complete synchronizations, that is synchronizations that are allowed on hidden nodes. For instance, in CCS-style synchronization just τ (and ϵ) are allowed on those nodes.

Conditions (i) and (ii) are from SAs. The former specifies that the result of an n-ary synchronization does not depend on the order in which actions are synchronized. The latter specifies that non ϵ actions can not disappear giving ϵ. Condition (iii) specifies that synchronization with ϵ, if allowed, just propagates the other action. Condition (iv) assures that all the edges can stay idle on any node. Conditions (v) and (vi) state that mobility patterns are associative and commutative, extending condition (i) to the mobility part. Finally, condition (vii) 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. In particular, this guarantees the existence of h, k in the definition of function par above.

We now characterize the effects of the synchronization specified by a SAM S on a n-uple of actions $\langle \langle a_1, \boldsymbol{y}_1 \rangle, \ldots, \langle a_n, \boldsymbol{y}_n \rangle \rangle$. The effects of the synchronization are an action c_n with a tuple of parameters $\boldsymbol{w}_n$ and a substitution ρ_n. We use $\mathrm{eqn}(\{t_1/x_1, \ldots, t_m/x_m\})$ to denote $\{t_1 = x_1, \ldots, t_m = x_m\}$.

Definition 3.2 (Effects of a synchronization) *The* effects of a synchronization *are computed by induction on the number n of actions.*

$n = 1$) $c_1 = a_1$, $\boldsymbol{w}_1 = \boldsymbol{y}_1$, $\rho_1 = \mathrm{id}$.

Inductive case) Let c_n, $\boldsymbol{w}_n$ and ρ_n be the effects of the synchronization among the first n actions. Then:

$c_{n+1} = c_n \bullet a_{n+1}$,
$\rho_{n+1} = \mathrm{mgu}(\mathrm{eqn}(\rho_n) \cup \mathrm{eqn}(\mathrm{sub}(S, \langle c_n, \boldsymbol{w}_n\rangle, \langle a_{n+1}, \boldsymbol{y}_{n+1}\rangle)))$,
$\boldsymbol{w}_{n+1} = \mathrm{par}(S, \langle c_n, \boldsymbol{w}_n\rangle, \langle a_{n+1}, \boldsymbol{y}_{n+1}\rangle)\rho_{n+1}$.

Conditions (i), (v), (vi) in Definition 3.1 ensure that the result is independent w.r.t. the order of $a_1, \ldots, a_n$.

We present now some simple SAMs which can be used as building blocks for more complex ones, highlighting the technical aspects of the formalization of a synchronization model as SAM. We just write the cases where $\bullet$ is defined. We also skip cases that can be derived by commutativity. Furthermore, in the examples, unless explicitly stated, we use $\mathrm{mob}_{a,b} = MP_{a,b}$. The following SAMs use the minimal number of actions necessary to model a synchronization of the chosen type, but sets of actions sharing just ϵ can be merged in a unique SAM allowing different policies. In this case the action performed chooses the protocol to be used, since it can interact only with other actions from the same group. An example of this kind is presented in § 5.

Example 3.3 (Mutual exclusion SAM)
The mutual exclusion SAM is defined by:

- $Fin = Act = \{a, \epsilon\}$;
- $\lambda \bullet \epsilon = \lambda$ *for each* $\lambda \in Act$.

Mutual exclusion ensures that in each transition at most one non ϵ action can be performed on each node. Synchronization of a with ϵ is necessary to allow transitions when more than one component is attached to the node. The SAM obtained by removing this synchronization allows to detect if an edge is the only one attached to a node, and it is attached just one time to it.

Example 3.4 (Milner SAM) *The Milner SAM is defined by:*

- $Act = \{a, \overline{a}, \tau, \epsilon\}$ *with* $\mathrm{ar}(a) = \mathrm{ar}(\overline{a})$ *and* $\mathrm{ar}(\tau) = 0$;
- $a \bullet \overline{a} = \tau$, $\lambda \bullet \epsilon = \lambda$ *for each* $\lambda \in Act$;
- $Fin = \{\tau, \epsilon\}$.

The Milner SAM, so called since it is inspired by π-calculus synchronization, models message passing, where actions a and $\overline{a}$ are input and output respectively and τ stands for a complete message exchange. During synchronization corresponding parameters are merged. Technically this is a refinement of the mutual exclusion SAM, in fact mutual exclusion is imposed between different synchronizations. Having just τ and ϵ in Fin ensures that on a hidden node x either nothing happens or a complete message exchange is performed. Note that a SAM that uses many actions, all interacting using the Milner protocol can be built, and many variations are possible. For instance, the desired possibilities of input-output interactions can be specified, e.g., allowing an input to interact with all the outputs in a given set.

We present now an extension of Milner SAM where communication has to be

authorized by a particular action ok, thus allowing a simple form of traffic control. Thus a τ is here obtained as a result of a synchronization among a, $\overline{a}$ and ok. We consider the simpler case of actions and coactions having arity 1.

Example 3.5 (Controlled Milner SAM) *The controlled Milner SAM is defined by:*

- $Act = \{a, \overline{a}, ok, (a, \overline{a}), (a, ok), (\overline{a}, ok), \tau, \epsilon\}$ *with* $\mathrm{ar}(\lambda) = 1$ *for each* $\lambda \in Act \setminus \{ok, (a, \overline{a}), \tau, \epsilon\}$, $\mathrm{ar}((a, \overline{a})) = 2$ *and* $\mathrm{ar}(\lambda') = 0$ *for each* $\lambda \in \{ok, \tau, \epsilon\}$;
- $a \bullet \overline{a} = (a, \overline{a})$ *with* $\mathrm{mob}_{a,\overline{a}}([1,1]) = 1$, $\mathrm{mob}_{a,\overline{a}}([2,1]) = 2$, $(a, \overline{a}) \bullet ok = \tau$ *with* $\mathrm{mob}_{(a,\overline{a}),ok}([1,x]) = 1$ *for each* $x \in \{1, 2\}$, $a \bullet ok = (a, ok)$, $\overline{a} \bullet ok = (\overline{a}, ok)$, $(a, ok) \bullet \overline{a} = \tau$, $(\overline{a}, ok) \bullet a = \tau$, $\lambda \bullet \epsilon = \lambda$ *for each* $\lambda \in Act$;
- $Fin = \{\tau, \epsilon\}$.

We have used here a technical trick: actions $(a, \overline{a})$, (a, ok) and $(\overline{a}, ok)$ are generally not used in productions, but they are used as intermediate results in the computation of the full synchronization. Note that here mobility patterns are not always specified by MP.

Example 3.6 (Broadcast SAM) *The broadcast SAM is defined by:*

- $Act = \{a, \overline{a}, \epsilon\}$ *with* $\mathrm{ar}(a) = \mathrm{ar}(\overline{a})$;
- $a \bullet \overline{a} = \overline{a}$, $a \bullet a = a$, $\epsilon \bullet \epsilon = \epsilon$;
- $Fin = \{\overline{a}, \epsilon\}$.

The broadcast SAM models secure broadcast, where one component performs an output and all the others perform input. Notice that here reaction with ϵ is not allowed, and this requires all the components to participate in a non idle way to the synchronization. The requirement can be weakened by allowing some components to stay idle, thus obtaining multicast.

Example 3.7 (Multicast SAM) *The multicast SAM is defined by:*

- $Act = \{a, \overline{a}, \epsilon\}$ *with* $\mathrm{ar}(a) = \mathrm{ar}(\overline{a})$;
- $a \bullet \overline{a} = \overline{a}$, $a \bullet a = a$, $\lambda \bullet \epsilon = \lambda$ *for each* $\lambda \in Act$;
- $Fin = \{\overline{a}, \epsilon\}$.

To clarify Definition 3.2 we show here the effects of the synchronization of a tuple of actions $\langle\langle a_1, \boldsymbol{y}_1\rangle, \ldots, \langle a_n, \boldsymbol{y}_n\rangle\rangle$ according to multicast SAM. The synchronization is allowed provided that at most one action is $\overline{a}$. On a hidden node exactly one action must be $\overline{a}$. Also, ρ is an mgu of $\{\boldsymbol{y}_{i_1} = \boldsymbol{y}_{i_2} = \cdots = \boldsymbol{y}_{i_m}\}$ where $\{i_1, \ldots, i_m\}$ are the indexes of the non ϵ actions. Finally, $\boldsymbol{w} = \boldsymbol{y}_{i_1}\rho$.

4 Deriving transitions from productions

In this section we present an algorithm to derive all the transitions starting from a graph G (without isolated nodes) specified by a set of productions $\mathcal{P}$ using a SAM S. All the actions used in $\mathcal{P}$ are required to belong to Act.

The steps of the algorithm are described below.

(i) For each edge e a production $P_e = L_e \xrightarrow{\Lambda_e,\mathrm{id}} R_e$ is chosen, in such a way that there exists an idempotent substitution $\sigma_e : \Gamma_{L_e} \to \Gamma_{L_e}$ such that $L_e\sigma_e$ is the subgraph of G composed by the edge e and the attached nodes. Essentially L_e is equal to the desired subgraph, but if e is attached many times to the same node x then all the occurrences of x but one are renamed in L_e using fresh names: σ_e performs the inverse substitution. Furthermore nodes created by the productions (i.e., in Λ_e but not in L_e) must be fresh.

(ii) For each node x, all the actions performed by productions P_e on nodes y such that $y\sigma_e = x$ are instantiated by applying σ_e to their tuples of parameters and then composed, producing an action c_x with parameters $\boldsymbol{w}_x$ and a substitution ρ_x.

(iii) If there is at least a node x for which the above action composition is not defined, or $x \notin \Gamma_G$ but $c_x \notin Fin$, then no transition can be derived for this choice of productions.

(iv) A global substitution ρ is defined as the composition of all the substitutions ρ_x, that is $\rho = \mathrm{mgu}\{\bigcup_{x\in N} \mathrm{eqn}(\rho_x)\}$. Among the possible mgus we choose one where nodes in Γ_G are taken as representatives of their equivalence classes whenever possible.

(v) Λ maps each node x in Γ_G to the pair $\langle c_x, \boldsymbol{w}_x\rho\rangle$.

(vi) π is the restriction of ρ to the nodes in Γ_G.

(vii) The final graph is obtained as follows:
- a graph is obtained by merging the instances $R_e\sigma_e$ of the RHSs of all the productions P_e (choosing different representatives for hidden nodes and edges in different RHSs);
- the substitution ρ is applied to the graph;
- only nodes in $\Gamma_G \cup \mathrm{n}(\Lambda)$ that occur in the resulting graph are kept in the interface;
- isolated nodes are deleted.

We will not state here a formal theorem relating our semantics with the one presented in [11], since the present semantics formalizes a more refined approach to PSHR w.r.t. the older one available in [11]. In fact, the two approaches allow slightly different classes of SAMs. We will however discuss informally the differences between the two semantics.

In [11] graphs are represented as syntactic judgments $\Gamma \vdash T$ where T is a term and Γ is the set of nodes in the interface (corresponding to Γ_G here). The term T is built using constants for edges and the empty graph, and operators for composing graphs (merging common nodes) and hiding nodes. Term T is considered up to a structural congruence that abstracts from the order of edges and of hidings and allows α-conversion of nodes. Edges are not explicitly named: just the labels are considered. Judgments up to structural congruence can be interpreted into graphs, obtaining a bijective correspondence.

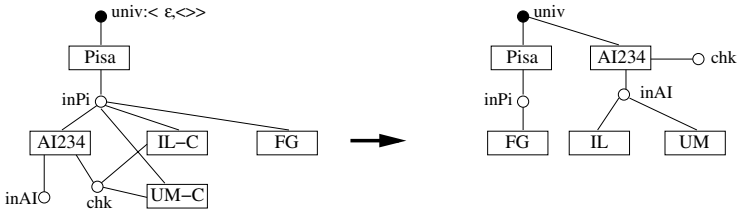

Fig. 1. A sample transition.

From a dynamic point of view the main difference between the semantics presented here and the standard one is that here isolated nodes are forbidden in the starting graph and removed from the result, while they are allowed in the standard one. This is not an important restriction since isolated nodes can not influence the other parts of the graph. They are actually needed in the standard semantics for the internal steps of the derivation of some transitions. In our case it is enough to allow them in productions. This difference allows to remove the component *Init*, used in [11], from SAM definition. Also, the standard semantics allows a non identity substitution π also in productions, but this is superfluous since the same effect can be obtained by synchronizing two actions on a hidden node. However this feature can be added also to our semantics. If we restrict our attention to SAMs that can be specified in both the frameworks (and we find a suitable set *Init* for [11]-style SAMs), and to productions having just id as node substitution, then the two semantics are equivalent up to isolated nodes (and actions performed on them).

5 The airport case study

We show here how the approach described above can be applied to the airport case study [2] of FET-GC project AGILE [1]. Since we are not aiming at tackling the whole case study, but just at showing how PSHR can be applied, we will do some simplifications. For a more complete approach to this modelling problem see [4].

The airport case study concentrates on modelling planes landing and taking off at airports, with passengers boarding the planes. We model entities (which are classes in UML class diagrams [13]) as edges with attributes modelled as nodes. In particular, we have edges for airports, planes and passengers. In this example, the first connection of each edge represents the attribute AtLoc, proposed in an extension of UML diagrams with mobility [5]. The value of attribute AtLoc represents the location containing a mobile object. Also, objects that are locations such as airports and planes have the dual attribute Containing. Furthermore, planes have an attribute CheckedIn, whose value is the set of passengers that have already checked in for next flight. Passengers that have already checked in have the dual attribute. A simple graph modelling a system of this kind is the left graph in Figure 1, featuring one airport (*Pisa*) located in the universe (*univ*), a plane (*AI*234) in the airport and three passengers, two which have already checked in (*IL-C*, *UM-C*), and one which has not (*FG*). We represent edges as rectangles containing the label and connected to bullets representing nodes. Bullets are solid for nodes in the

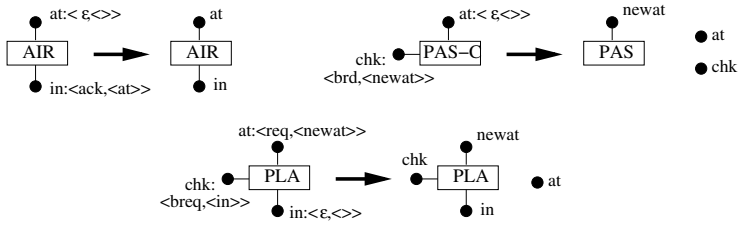

Fig. 2. Productions for the example.

interface and empty otherwise.

We want to specify a transition that models the boarding of all the passengers who have checked in and the take off of the plane. This transition requires multiple checks and reconfigurations: essentially all the passengers who have checked in must move (changing their location), and the airport must allow the plane to take off. The plane must change its location too.

This is modelled by the productions in Figure 2 (which are schemas drawn for generic labels AIR, PLA, $PAS\text{-}C$ and PAS for airports, planes, checked in passengers and not checked in passengers respectively), where Λ is represented by decorating each node with the corresponding action.

We have now to specify the SAM S that we want to use. Actions ack and req have to synchronize using Milner synchronization, since they model a message exchange between the airport and the plane allowing the take off, while actions $breq$ and brd have to synchronize using broadcast synchronization with $breq$ as output action, since all the checked in passengers have to board. Thus we can build the wanted SAM using the Milner SAM and the broadcast SAM as building blocks. The resulting SAM is defined by:

- $Act = \{req, ack, breq, brd, \tau, \epsilon\}$ with $\mathrm{ar}(\lambda) = 1$ for each $\lambda \in Act \setminus \{\tau, \epsilon\}$ and $\mathrm{ar}(\tau) = 0$;
- $req \bullet ack = \tau$, $breq \bullet brd = breq$, $brd \bullet brd = brd$, $\lambda \bullet \epsilon = \lambda$ for each $\lambda \in \{req, ack, \tau, \epsilon\}$;
- $Fin = \{\tau, breq, \epsilon\}$.

We can thus derive the transition in Figure 1. Let us see how the different steps of the algorithm are performed.

(i) For each edge but FG the corresponding production in Figure 1 is used, for edge FG an idle production is used. For nodes in the LHSs the names in the graph can be used, since no edge is attached two times to the same node. For new nodes (all called $newat$ in the figure) different names must be chosen. To this end we add the label of the corresponding edge to the nodes created by passenger edges.

(ii) Let us consider node $inPi$ as example. The actions performed on it are $\langle ack, \langle univ \rangle\rangle, \langle req, \langle newat \rangle\rangle, \langle \epsilon, \langle\rangle\rangle, \langle \epsilon, \langle\rangle\rangle, \langle \epsilon, \langle\rangle\rangle$. These can be composed producing $\rho_{inPi} = \{univ/newat\}$ and action $\langle \tau, \langle\rangle\rangle$.

(iii) The transition is allowed since all the compositions are defined and for nodes

different from *univ* the resulting action is in Fin.

(iv) The substitution ρ is $\{univ/newat, inPi/newat_{IL-C}, inPi/newat_{UM-C}\}$.

(v) Λ maps just *univ* to $\langle \epsilon, \langle\rangle \rangle$.

(vi) π is the identity substitution.

(vii) The final graph is obtained from the union of the RHSs, applying substitution ρ and leaving only *univ* in the interface.

Notably, when a suitable SAM is chosen for synchronization, the implementation of the communication protocol becomes trivial. In [4] instead just a simple binary synchronization is used, thus a complex procedure is required to implement broadcast. In particular, this adds to the model of the system a subgraph used for synchronization purposes which does not correspond to any entity in the real system. Our choice allows models at a more abstract level, as suited for modelling complex systems.

6 Conclusion and future work

We have provided a more direct characterization of the behavior of a SAM and of the transitions allowed by PSHR w.r.t. [11]. We think that this is useful to make PSHR more usable. The result applies also to most of the SHR frameworks in the literature, which are instances of PSHR with a suitable SAM. Our approach can be also straightforwardly extended to deal with nondeterministic synchronizations and the use of many SAMs inside the same graph as presented in [12].

As future work we want to formalize different forms of SAM composition using categorical tools and analyze the observational semantics of SHR systems.

References

[1] AGILE: Architectures for mobility. http://www.pst.informatik.uni-muenchen.de/projekte/agile/.

[2] L. F. Andrade et al. AGILE: Software architecture for mobility. In *Proc. of WADT'02*, volume 2755 of *LNCS*, pages 1–33. Springer, 2002.

[3] Architecture description languages. http://www.sei.cmu.edu/architecture/adl.html.

[4] P. Baldan, A. Corradini, and F. Gadducci. Specifying and verifying UML activity diagrams via graph transformation. In *Proc. of Global Computing 2004*, volume 3267 of *LNCS*, pages 18–33. Springer, 2004.

[5] H. Baumeister, N. Koch, P. Kosiuczenko, and M. Wirsing. Extending activity diagrams to model mobile systems. In *Proc. of NetObjectDays'02*, volume 2591 of *LNCS*, pages 278–293. Springer, 2002.

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

[7] H. Ehrig, M. Pfender, and H. J. Schneider. Graph grammars: an algebraic approach. In *Proc. of SWAT '73*, pages 167–180, IEEE, 1973.

[8] 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, 2001.

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

[10] I. Lanese. *Synchronization strategies for global computing models*. PhD thesis, Computer Science Department, University of Pisa, Pisa, Italy, 2006.

[11] 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 *ENTCS*, pages 43–60. Elsevier, 2004.

[12] I. Lanese and E. Tuosto. Synchronized hyperedge replacement for heterogeneous systems. In *Proc. of COORDINATION 2005*, volume 3454 of *LNCS*, pages 220–235. Springer, 2005.

[13] J. Rumbaugh, I. Jacobson, and G. Book. *The Unified Modeling Language Reference Manual*. Addison Wesley, 1999.

[14] G. Winskel. Synchronization trees. *TCS*, 34:33–82, 1984.