Tarzan: Communicating and Moving in Wireless Jungles

Emilio Tuosto

Dipartimento di Informatica, Via F. Buonarroti 2, 56127 Pisa – Italy

Abstract

Wireless networks allow portable/mobile devices (and the related applications) to communicate each other with radio or infrared signals. Computations on those networks are very sensible to many physical variables (e.g., energy consumption, distance, geographic topology, etc.). We propose Tarzan, a framework based on graph rewriting that easily allow one to express many quantitative aspects of wireless networks. Apart from the expressiveness issue, the main advantage of Tarzan is its formal semantics that can be exploited for specifying applications, routing algorithms or coordination aspects of wireless devices considering realistic physical limitations.

1 Introduction

Networks of wireless devices present peculiarities that result of minor or no importance in wired networks. Among others, dynamism of network topology, energy constraints and transmitting capacity are of great moment. Topology of wireless networks is highly dynamic because nodes in general are portable computational devices (e.g. mobile phones, PDA's, laptops,...). Nodes can asynchronously disappear because of battery consumption. Apart from the impact on interactions among devices and dynamism of the topology, this is an important issue also because wireless networks typically are *peer-to-peer* network where any node can act as a gateway. Wireless devices have limited transmitting capacity. Therefore, communication is not always guaranteed; moreover, the external environment might cause interferences or interdict communications.

Summing up, wireless networks raises many issues that intrinsically involve several quantitative aspects that must be taken into account in the computation. In general, those aspects are considered separately and studied within a

⋆ This work has been supported by the european EU-FET project **PROFUNDIS** IST-2001-33100 and the MIUR project **SP4** "Architetture Software ad Alta Qualità di Servizio per Global Computing su Cooperative Wide Area Networks".

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

mathematical setting that models the physical characteristics of devices and external environment with respect to routing algorithms. As far as we know, a formal framework for expressing the above discussed phenomena in a uniform framework is still lacking.

Graph-based techniques can be adopted for modelling inter-networking systems. Indeed, edges can be used to represent components of the system, while nodes model the synchronisation ports of the components. Two (or more) edges sharing a node represent components connected on a communication port; adjacent edges can synchronise. Graphs can be opportunely synchronised by means of an operational semantics. Graph rewriting based on edge replacement and synchronisation was introduced in [2,5] and related to distributed constraint satisfaction problems in [14]. Mobility has been recently considered in [9,10,6,16] where *synchronised hyperedge replacement* (SHR) has been investigated.

We consider purely local graphs synchronisation obtained by combining graph rewriting and constraint solving. The intuitive idea is that local rewritings depends on the outcome of a constraint satisfaction algorithm. Our framework is a simplification of the SHR approach in [6] and permits interconnection modification. Nodes can be exchanged during synchronisations, hence constraint solving must encompass unification in order to fuse node. This amounts to *mobility* of components, that dynamically can change their connections. On the top of the SHR approach, we describe Tarzan, a semantic model suitable for dealing with those aspects of wireless networks. Interestingly, graph rewritings seem to be suitable for describing some phenomena that traditional models of distributed computations cannot apparently capture (at least in a simple manner, see the discussion in Sections 2 and 6). In particular, communication interferences can be simply expressed in a formal way accounting for formal reasoning on wireless networks. This seem to be an interesting and not completely explored field; indeed, quoting [7]:

> *For ad-hoc networks that share the same spectrum, new methods of cooperation are required to permit coexistence. Such methods are difficult to research without real-world channel models and simulation methodologies; there is still fundamental work to be done in this area [13].*

Apart from a theoretical interest, we suggest Tarzan as a formal language for expressing computations in wireless networks and modelling the physical environment. As will be more clear later (Section 5) the environment resembles a luxuriant vegetation of a tropical jungle with lianas where Tarzan (a wireless device) swings and yells looking for Jane (his partner). Even though we do not address wireless protocols design, this approach can also help designers in correctly specifying their protocols.

Structure of the paper. Section 2 briefly discusses the difficulties that traditional models of distributed computations have when wireless communications are under consideration. Section 3 gives an intuitive interpretation

of SHR. Section 4 reports the formal definitions of hypergraphs and productions; moreover, an intuitive operational interpretation of SHR is given in Section 4.3 by means of an example. In order to maintain the presentation as simple as possible, we relegated the formal definition of SHR in Appendix A. Productions for Tarzan are defined in Section 5. Final remarks are in Section 6.

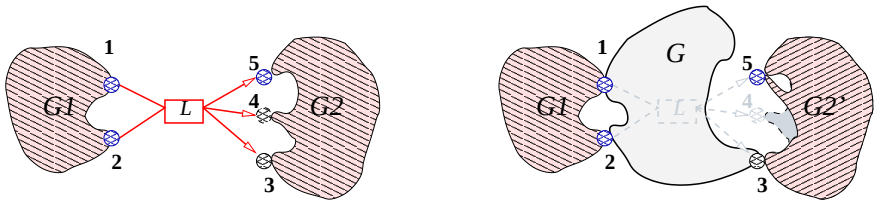
2 Wireless Networks and Traditional Models

Differently from wired networks, wireless devices can interact each other without a pre-existing communication infrastructure. Indeed, they are typically equipped with both radio and infrared transmitters/receivers.

A semantical framework for modelling and reasoning on wireless systems is inherently much more difficult to define than a framework for "wired-systems". The main reason being that the communication infrastructure does not permit to individuate the position of components by their name (as discussed in Section 1). This is not only problematic for modelling interaction of components, but also because the semantic model should encompass the concept of *distance*. In fact, the framework should model the topology (together with its geometrical/physical structure), the structure of the network *and* distances between two components, their transmitting capacity, etc.

Traditional frameworks like Ambient [1] or Klaim [4] do not seem able to capture the great part of these aspects in an elegant manner. An Ambient system is the parallel composition of *ambients* and that are terms of the form $a[P_1|\ldots|P_n|b_1[\ldots]|\ldots|b_m[\ldots]]$ (a, b_i's are names and P_j's are subsystems). Hence the topology of an Ambient system if a forest of trees. Processes send messages "in the air", receive them and guide their surrounding ambient through the system topology. The condition for a message being received is that both the message and the receiver must be inside the same ambient. However, this "vicinity" condition does not encompass any distance concept because the parallel composition operator $|$ is commutative, therefore $a[P|Q|R]$ is equivalent to $a[P|R|Q]$. In Klaim systems are parallel composition of *nodes* $s :: P_1|\ldots|P_n$, where s is the name of the node, P_i's are either processes running at s or tuple of data allocated at s. Processes can generate tuples, input them via pattern-matching and roam through nodes. Klaim messages "float in the air" of s, however nesting of nodes is not allowed, hence Klaim nets resemble flat graphs.

Recently, Klaim has been equipped with linguistic mechanisms for controlling links among nodes [3]. Link creation and remotion can be programmed and the language can also express link features that depend from the applications. Indeed, distance between nodes can be easily expressed in this context. However, neither Ambient nor Klaim can easily model an interesting phenomenon of wireless communications, namely the interference on communications caused by *third party* movements. Indeed, the communication between two wireless devices D_1 and D_2 can be disturbed by another component



(a) A hyperedge L (b) Replacement of L with G

Fig. 1. Hyperedge replacement

(not necessarily a device) that moves trough the space separating them. This phenomenon is difficult to capture in traditional frameworks because even if a link encompass the distance between nodes, it is under the control of the connected nodes and a third entity cannot "break" or modify it.

3 Hypergraphs: an Overview

Graph grammars [15] have been proposed as an extension of grammars of strings. Similarly to grammars of strings, *productions of graph grammars* have the form $L \longrightarrow R$ where L and R are graphs and specify how graphs can be rewritten. Conditions on $L \longrightarrow R$ allow one to introduce the concept of "context-freeness", e.g. productions with a left-hand-side (lhs) which is either a node or an edge confer a "context-free" flavour to graph grammars; indeed, such productions do not consider the "surroundings" of their lhs.

Hypergraphs are graphs made of nodes and *hyperedges*, a generalisation of edges; a hyperedge may be thought of as being an edge connecting more than two nodes. An edge in a traditional graph can be intuitively thought of as representing a binary relation between two nodes, while a hyperedge represents a relationships among many nodes.

We will describe a rewriting mechanism based on synchronised hyperedge replacement; indeed, such mechanism will be exploited to constraining graph rewriting. Figure 1 aims at giving an intuition of hyperedge replacement. The hyperedge L in Figure 1(a) is connected to $G1$ and $G2$ through its *external nodes* (or *attachment points*) $\mathbf{1}, \ldots, \mathbf{5}$: Figure 1(b) represents the hypergraph obtained by replacing L with hypergraph G. The dashed gray part of the Figure 1(b) represents the initial situation and disappears after L has been replaced by G; notice that G_1 is not involved in the rewriting. Moreover, new nodes can appear (all nodes in G but **1** and **2** are new nodes generated by the transition) and some nodes can be "fused" after the transition (in Figure 1(b) node **4** is fused with **5**). As will be shown later, this amounts to *mobility* of components, that dynamically can change their connections. In fact, notice that, in Figure 1(b), the part of $G2$ that was connected to node **4** corresponds, in Figure 1(a), to the part of $G2'$ connected to node **5**.

4 Hypergraphs

In the following we consider fixed a set of nodes $\mathcal{N}$ and a set of labels $\mathcal{L}$ ranked by natural numbers; $L : n$ denotes a label $L \in \mathcal{L}$ with rank n.

Hypergraphs are built out from *hyperedges* and nodes. A *hyperedge* (or simply an *edge*) labelled by L may connect as many nodes as its rank. We write $L(x_1, ..., x_n)$ to indicate an edge labelled L connected to the attachment nodes x_1,..., x_n. For instance, hyperedge L in Figure 1(a) is written as $L(\mathbf{1}, \ldots, \mathbf{5})$. Wires from nodes $\mathbf{1}, \ldots, \mathbf{5}$ to L are the *attachment point* of L and are called *tentacles*.

Formally, hypergraphs are *syntactic judgements*:

Definition 4.1 $\Gamma \vdash G$ is a *syntactic judgement* iff

- $\Gamma \subseteq \mathcal{N}$ is a finite set of nodes and
- G is a term generated by the following grammar

$$G ::= nil \mid L(x_1, \ldots, x_n) \mid G \mid G,$$

where $x_1, \ldots, x_n$ are nodes and $L \in \mathcal{L}$ is such that $L : n$. We call terms G *hypergraph terms* and let $\mathrm{n}(G)$ denote the set of nodes that appear in G.

Productions in Definition 4.1 permits generating the empty graph (represented by *nil*), single edges (using $L(x_1, \ldots, x_n)$) and composing terms in parallel (via $G \mid G$). Hereafter, we use 'graph' as a synonym of 'hypergraph' and omit curly brackets in judgements writing $x_1, ..., x_n \vdash G$ instead of $\{x_1, ..., x_n\} \vdash G$.

4.1 Hypergraph Synchronisation

SHR is obtained by graph rewriting combined with constraint solving. More specifically, we use *context-free* productions labelled with actions useful for coordinating the simultaneous application of productions. Coordinated rewriting allows the propagation of synchronisation all over the graph where productions are applied. Determining the productions to synchronise at a given stage corresponds to solving a distributed constraint satisfaction problem [14].

The main feature of the SHR approach is that an hyperedge can be replaced when the conditions it imposes on its external nodes are in accordance with the conditions imposed by adjacent hyperedges and with the adopted synchronisation policy. Replacements can generate new nodes, exchange nodes and fuse them.

Observation 4.1 *Each edge rewriting must synchronise its actions with one or more of its adjacent edges. Depending on the chosen synchronisation algebra, the number of edges can vary.*

The main and well studied synchronisation mechanisms are á la *Hoare (CSP [11], where $S(a, a) = a$) and* á la *Milner (CCS [12], where $S(a, \bar{a}) = \tau$) synchronisations [17] (conventionally, one can think of $\bar{a}$ as an output action,*

while a denotes an input action). The former requires that all participants of a synchronisation perform the same action, while the latter takes two partner to synchronise through complementary actions. Hereafter, we consider synchronisations á la *Milner.*

In the following, *Cons* denotes the set of actions, namely, the conditions imposed on the external nodes of hyperedges for constraining graph rewritings. Since we use Milner synchronisations, *Cons* also has two further ingredients:

- a complementation operation $\bar{\cdot} : \mathit{Cons} \to \mathit{Cons}$ such that for any $a \in \mathit{Cons}$, $\bar{\bar{a}} = a$;
- a distinguished *silent* action τ that denotes synchronisations.

Moreover, we assume that any label $a \in \mathit{Cons}$ has an *arity*; we let $|\cdot| : \mathit{Cons} \to \omega$ be the arity function on *Cons* and, for any $a \in \mathit{Cons}$, $|a| = |\bar{a}|$.

4.2 Productions

A *graph rewriting system*, $\mathcal{G} = \langle \Gamma_0 \vdash G_0, \mathcal{P} \rangle$, consists of an initial graph $\Gamma_0 \vdash G_0$ and a set of *productions*. Before defining productions, we need an auxiliary definition.

Definition 4.2 [Fusion substitution] Let X be a set of nodes. A fusion substitution on X is a function $\pi : X \to X$ such that

$$\forall x, y \in X. \pi(x) = y \Rightarrow \pi(y) = y.$$

A fusion substitution $\pi : X \to X$ induces an equivalence relation $\simeq_\pi$ over X defined as $x \simeq_\pi y \stackrel{\text{def}}{\iff} \pi(x) = \pi(y)$. Equivalence $\simeq_\pi$ partitions X into equivalence classes and each node $x \in X$ is mapped to the *representative element* $\pi(x)$ of its class, namely the unique element y such that $\pi(y) = y$.

Definition 4.3 [Production] Let $X \subseteq \mathcal{N}$ be the set $\{x_1, \ldots, x_n\}$ and $L \in \mathcal{L}$ such that $L : n$. A *production* is a transition of the form

$$X \vdash L(x_1, \ldots, x_n) \xrightarrow[\pi]{\Lambda} \Gamma \vdash G, \tag{1}$$

where $\mathrm{n}(G) \subseteq \Gamma$ and

- $\Lambda \subseteq X \times \mathit{Cons} \times \mathcal{N}^*$ is a *set of constraints* which are triples $(x,\ a,\ \boldsymbol{y})$ such that the length of $\boldsymbol{y}$ is $|a|$. Given Λ, we say that $\boldsymbol{y}$ are the *nodes of the constraint* $(x, a, \boldsymbol{y}) \in \Lambda$ and $\Lambda(x)$ indicates $(a, \boldsymbol{y})$, while $\Lambda(x) \uparrow$ states that Λ does not imposes constraints on x (i.e. $(x', a, \boldsymbol{y}) \in \Lambda$ implies $x \neq x'$). We let $\mathrm{n}(\Lambda)$ denote the union of the nodes of the constraints in Λ.
- $\pi : X \to X$ is a fusion substitution such that $\mathrm{n}(\Lambda) \cap X \subseteq \pi(X)$ (namely, the external nodes used in the synchronization must be representative elements according to $\simeq_\pi$);
- $\Gamma = \pi(X) \cup (\mathrm{n}(\Lambda) \setminus X)$.

Nodes $x_1, \ldots, x_n$ are the attachment nodes of L to the surrounding environment, namely L can share them with other edges. Productions specify the constraints that the environment must satisfy in order to replace edges. Such constraints are imposed by Λ on the set X of external nodes of L; arities of actions must be equal to the number of nodes of the constraint. Λ associates actions and sequences of nodes to (some of the) external nodes of L. Intuitively, actions associated to attachment nodes constrain the possible rewritings of a graph; indeed, production (1) can be applied only if the actions on the external nodes synchronize with actions imposed by the productions of adjacent edges according to the adopted synchronization policy. If $(x, a, \boldsymbol{y}) \in \Lambda$ then L can synchronize with edges in the environment that have a tentacle connected to x and satisfy condition a.

Let us now consider the structure of the right hand side of production (1). Γ consists of the nodes which are image of $x_1, \ldots, x_n$ through π and the new nodes used in the synchronization, namely those nodes that appear in Λ and are not in X. In general, G may be any graph (provided that $\mathrm{n}(G) \subseteq \Gamma$).

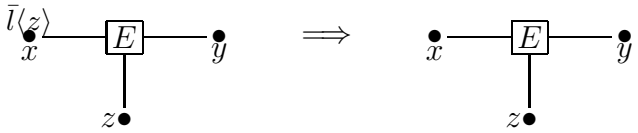
In the following we write $X \vdash L(x_1, \ldots, x_n) \xrightarrow{\Lambda} \Gamma \vdash G$, for a prodution whose fusion substitution is the identity function.

Once constraints in Λ are satisfied, nodes must be coalesced according to fusion substitution π.

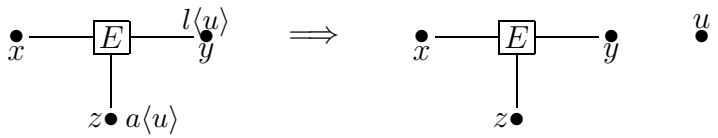
4.3 Synchronising productions

This section provides an example of application of SHR. We prefer to leave the presentation as simple as possible, therefore, we give an intuitive description of SHR-based rewritings. The formal machinery is reported in Appendix A. Instead of using the "textual" description of productions, we state them directly in terms of graphs. The following productions represent how an edge can migrate. We will exploit them in the design of the Tarzan framework.

We consider a simple example where two edges, $E : 3$ and $D : 1$ interact. The production



states that an edge $E(x, y, z)$ synchronises on node x with an l-action and communicates its third node z. After the synchronisation, E remains unchanged. The second production for E-edges is



and states that $E(x, y, z)$ simultaneously synchronises on node y and z with

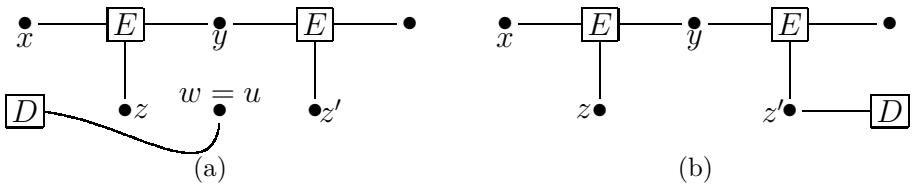
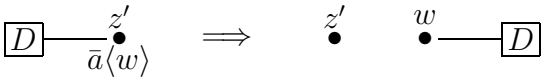


Fig. 2. Edge moving

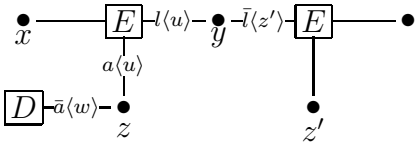
action l and a, respectively. In the rhs of the production the new node u appears. The intuition is that the node from an adjacent E-edge is received with l synchronisation and communicated to an edge connected to z.

Edge D has only one production



stating that, when $D(z')$ synchronises on z' with action a, then a new node w is received; after the synchronisation, the D-edge connected to z' is cancelled and a new instance of D-edge appears connected on the new node w.

Let us consider the following graph (tentacles have been annotated with the actions of the productions):



(2)

Rewritings of graph (2) are determined by the following steps:

(i) Individuate the adjacent tentacles labelled by complementary actions;

(ii) determine the synchronising productions;

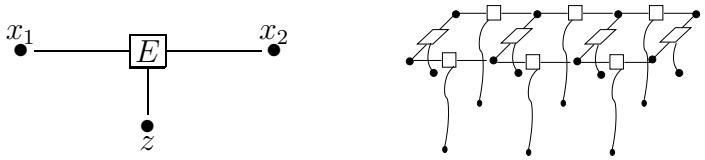
(iii) replace the (instance of) edges appearing in the lhs of synchronising productions with the corresponding rhs;

(iv) fuse those nodes that are equated by the synchronisations.

Let us apply the previous schema to graph (2). Step (i) is already done in (2); step (ii) is easy: The rightmost and the leftmost instances of E-edge are respectively rewritten by the first and second production of E-edges, while for edge D there is only one possible production. Step (iii) gives rise to the graph in Figure 2(a). Since the synchronisations impose the equations $w = u = z'$, step (iv) simply fuses w and z', determining the graph in Figure 2(b), where D is moved from z to z'.

The intuitive description of SHR given in this section suggests the following design style:

- Represent systems as a number of components;
- for each component, draw a hyperedge so that its external nodes are explicited (as in the reported example);



(a) Environment Edge

(b) Environment graph

Fig. 3. Environment edges and graphs

- then, on the external nodes, write the actions that will constraint the hyperedge rewriting;
- finally, the graph that will be substituted with the hyperedge is specified taking care that the external nodes of the graph satisfy the conditions of Definition 4.3.

In the following we will adopt this design style.

5 Tarzan

We exploits the graphical calculus reported in the previous sections to model both wireless devices *and* the "environment" where they run. Our framework, Tarzan, takes advantage of the declarative flavour of SHR since devices are *autonomous* while mobility is achieved by node fusion. We do not consider all the issues discussed before, but give hints on how they can be modelled in Tarzan.

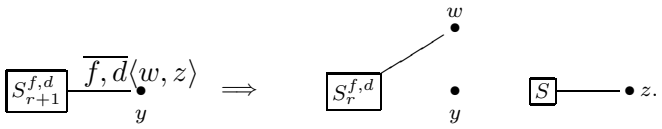
The environment is built by connecting *environment edges* $E : 3$ that have three tentacles and can be depicted as in Figure 3(a). Edge $E(x_1, x_2, z)$ can be thought of the atomic item that composes an ideal "ether" that allows signals to flow from x_1 to x_2 (and viceversa); node z is the node where edges representing wireless devices will be connected. As suggested by Figure 3(b), one can thought of the environment as a luxuriant vegetation of a tropical jungle with a lot of lianas where Tarzan (a wireless device) swings and yells looking for Jane (his partner).

Productions for signals

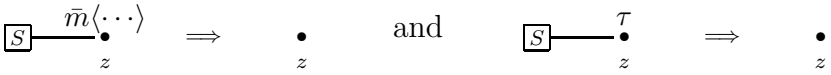
The main features of signals that we want to model are: (*i*) band frequency, (*ii*) intensity and (*iii*) direction. In general, other informations should be considered, for instance, signals carry data for routing or applications. However, we want only to define a framework for describing how signals propagate in a given environment. Considering more informative edges is a simple generalisation of the following definitions [1] .

[1] Here we do not consider broadcast signals that can be modelled at the cost of some complication with Milner synchronisation, whereas Hoare synchronisations could be more adapt.

We assume the existence of *signal edges* of arity 1 labelled by $S_r^{f,d}$, where f is a band frequency, d is a direction (commented later) and r is a natural number that abstracts the intensity of the signal. Productions for signal edges are very simple: As far as the intensity is not null, the signal tries to propagate along the assigned direction:



Only signals with non-null intensity can propagate. The intuition is that $S_{r+1}^{f,d}$ acquires from the environment the node w which is the next node along the direction d and the node z where a device is possibly connected. After the synchronisation, the weaker signal edge $S_r^{f,d}$ is attached to w and the *information edge* S accounts for interacting with the devices (possibly) encountered in that direction. The productions for S-edges are:

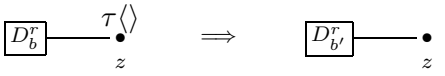


The first production considers the case where the information reach a device connected to z, while the second production states that the message can disappear without being received [2]. This models the case that no device is connected to z. Action m in general depends on the messages that must be exchanged. Here, we do not consider the informations that are relevant because they depend on the applications one wants to describe.

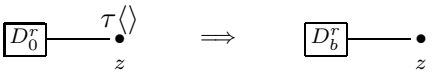
Productions for devices

Similarly to signal edges, device edges have arity 1. The main characteristics of devices we are interested in are: (i) the energy level, and (ii) the transmitting capacity. We consider constant the transmission power of a device [3], while the energy level is dealt with as done for the intensity of signal edges. Device edges are labelled by D_b^r; r is the *carrier sense* radius of the device, and b represents the residual energy of D (for simplicity we can assume that b is a positive integer).

We first consider productions for handling energy consumption:



(3)



(4)

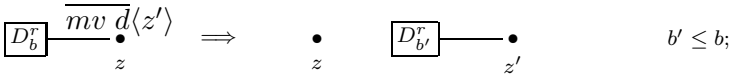
Production (3) states that the energy level of a device can non-deterministically decrease. Condition $b' = b$ captures the case of D representing a plugged in

[2] A similar production can be specified for $S_0^{f,d}$ if we want collect garbage signals.

[3] This is not always the case, but we want to keep this presentation as simple as possible. Simple variations of our productions can easily handle the general case.

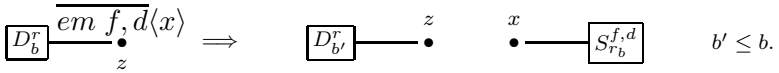
devices. Production (4) can appear strange at a first glance; it states that a discharged device suddenly acquires some energy. Basically, this models the fact that a device is turned on, at a given time.

Device mobility is modelled similarly to flow of signals:



D^r_b asks for the "next liane" along direction d where to jump to and, when found, the device moves. Notice that also in this case energy level might decrease.

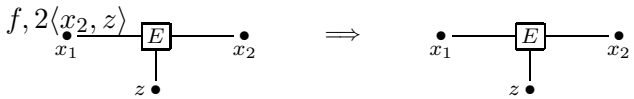
Last production deals with the emission of signals:



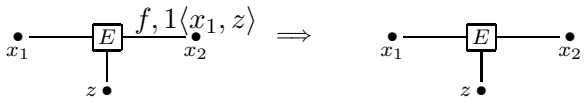
The intuition of the previous production is that device D^r_b attaches the signal-edge $S^{f,d}_{r_b}$ to the node x acquired from the environment edge. Such edge will propagate along direction d within a distance r_b (that, in general, depends on b) and will behave according to the productions defined for signal edges.

Productions for environment

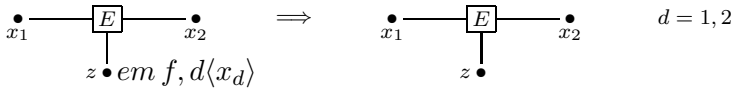
The environment is modelled by means of edges $E : 3$ whose main activity is the propagation of signal:



If E, on node x_1, is asked for the next node (from a signal edge), then it communicates node x_2 (as the next node) and z (as the liane where node S-edge must be connected). The dual situation takes place when on x_2 a signal wants to propagate toward x_1:



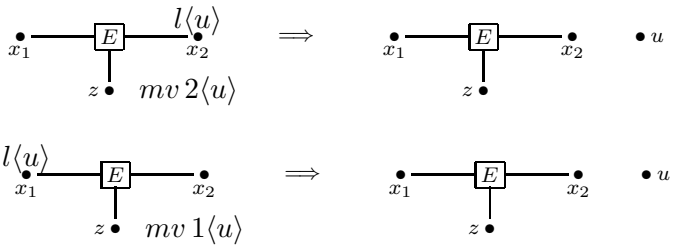
When a device on z wants to emit a signal, E must communicate x_1 or x_2 depending on the direction:



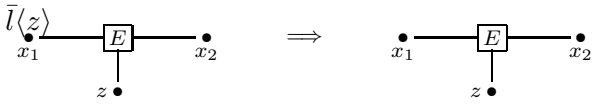
the environment node communicates x_1 or x_2, depending d.

Mobility of devices is more involved than signal propagation. Indeed, a device must "swing from liane to liane". Hence an environment edge must coordinate device action mv and ask to its neighbour environment (depending

on the direction it wants to move toward) for the next liane (action l):



The above productions perform such coordination; we comment only the first one, the second being analogous. If a device attached on z wants to move toward x_2, then the environment edge asks for the z-node of its neighbour. On reception, the device will attach itself on that node. Notice that, apart from the labels, this is exactly the same mechanism described in Section 4.3, indeed, in order to commit device movements, environment edges must perform the complementary actions of l:



(notice that the z-node of E is communicated). An analogous production is necessary for communicating z on x_2 when the device wants move toward x_1.

6 Final remarks

Modern inter-networked systems encompass several physical networks, multiple administration domains and a variety of possible users. Researches on global computing investigate the complex interactions taking place in inter-network computations. Many of those mainly focus on the *spatial structure* of systems with explicit localities, which model distributed computations and distributed resources and services composition. The main difference with respect to models of traditional distributed systems is *network awareness*: Localities are under programmer's control.

Traditional models of distributed systems assume that the addresses implicitly contains informations on the position of remote resources with respect to each component. Namely, the address of a resource implicitly describes how to reach it. A paradigmatic example of this addressing mechanism being IP-addressing. This is a strong assumption in wireless communicating systems and probably is one of the main difficulties for formally modelling them. A component that knows the address of a wireless device has no information on the position of that device.

We have shown how SHR can suitably represent those aspects of wireless communications (that seem difficult to be handled with more traditional approaches as discussed in Section 2). Our framework presents intrinsic peculiarities that have a precise counterpart in distributed programming (e.g. "declarativity as autonomy" and "node-fusion as mobility"). Moreover we specified Tarzan that extends the applicability of SHR to wireless communications. Tarzan is very flexible ("agile"). For instance, without any significant change to its basic mechanisms, it is possible to distinguish between radio and infrared (IR) devices; we must simply consider action label f as a metavariable ranging over $\{\text{radio}, \text{IR}\}$. Hence, we can also model "obstacles" that may be present in the environment. For instance a wall is simply an edge (connected to environment edges) having the same productions of E plus the additional condition that $f \neq \text{radio}$, which essentially states that a wall permits IR signals to traverse it but blocks radio waves.

All these peculiarities make Tarzan a expressive and extensible formal framework for modelling wireless systems or algorithm for wireless networks. The interesting feature of Tarzan is that routing algorithms can be programmed orthogonally to environment and signal propagation. Indeed, it is sufficient to specify productions like the last one for signal edges by supplying the routing information (informations like routing tables can be represented as suitable edge-labels attributes). As far as we know, the separation of concerns between routing algorithms and mobility or communication models is an innovative feature of our approach.

In this work we do not exploit the full expressiveness of SHR mechanism defined in [16,6]. Indeed, fusion substitutions are a powerful linguistic mechanism that permits merging communication points. (e.g., nodes of hypergraphs). This can be very useful for describing routing algorithm and coordinating activities of *adhoc networks*. For instance, a typical dynamic reconfiguration policy of adhoc networks is to collect devices of different subnets into a new net. Fusion substitutions can easily model this reconfiguration mechanism. We leave the formal definition for future investigations.

A final consideration is on the synchronisation policy adopted. Here we chose synchronisation *a lá* Milner, however, we plan to extend SHR for permitting both Milner and Hoare synchronisations. This is useful for a better handling of broadcasting. Indeed, Hoare synchronisations are a natural model for broadcasting communications and this would allow Tarzan signals to propagate in *any* direction.

Acknowledgement

The author thanks the anonymous referees for their useful comments and suggestions. The author is particularly grateful to Gianluigi Ferrari, Dan Hirsch and Alessandro Urpi for their useful comments and suggestions on a very preliminary version of this work.

References

[1] Cardelli, L. and A. Gordon, *Mobile ambients*, TCS: Theoretical Computer Science **240** (2000).

[2] Castellani, I. and U. Montanari, *Graph Grammars for Distributed Systems*, in: H. Ehrig, M. Nagl and G. Rozenberg, editors, *Proc. 2nd Int. Workshop on Graph-Grammars and Their Application to Computer Science*, Lecture Notes in Computer Science **153** (1983), pp. 20–38.

[3] De Nicola, R., 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 – Honoring Zohar Manna's 64th Birthday*, Lecture Notes in Computer Science, Springer-Verlag, 2003 pp. 436–479.

[4] De Nicola, R., G. Ferrari and R. Pugliese, *KLAIM: A kernel language for agents interaction and mobility*, IEEE Transactions on Software Engineering **24(5)** (1998), pp. 315–330.

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

[6] Ferrari, G., U. Montanari and E. Tuosto, *A LTS semantics of ambients via graph synchronization with mobility*, in: *7th Italian Conference on Theoretical Computer Science – ICTCS'01*, LNCS **2202** (2001), pp. 1 – 16.

[7] Giordano, S., "Handbook of Wireless Networks and Mobile Computing," John Wiley & Sons, 2001 pp. 325–343.

[8] Hirsch, D., "Graph Transformation Models for Software Architecture Styles," Ph.D. thesis, Departamento de Computación, Universidad de Buenos Aires (2003), http://www.di.unipi.it/~dhirsch/thesis.pdf.

[9] Hirsch, D., P. Inverardi and U. Montanari, *Reconfiguration of software architecture styles with name mobility*, in: A. Porto and G.-C. Roman, editors, *Coordination 2000*, LNCS **1906** (2000), pp. 148–163.

[10] Hirsch, D. and U. Montanari, *Synchronized hyperedge replacement with name mobility: A graphical calculus for name mobility*, in: *12th International Conference in Concurrency Theory (CONCUR 2001)*, LNCS **2154** (2001), pp. 121–136.

[11] Hoare, C., "Communicating Sequential Processes," Prentice-Hall, Englewood Cliffs, NJ, 1985, & 0-13-153289-8.

[12] Milner, R., "Communication and Concurrency," Printice Hall, 1989.

[13] Mobile Man Project, *Nfs tetherless t3 and beyond workshop*, Interim report (1998).

[14] Montanari, U. and F. Rossi, *Graph rewriting and constraint solving for modelling distributed systems with synchronization*, in: P. Ciancarini and C. Hankin, editors, *Proceedings of the First International Conference COORDINATION '96, Cesena, Italy*, LNCS **1061** (1996), pp. 12 – 27.

[15] Rozenberg, G., editor, "Handbook of Graph Grammars and Computing by Graph Transformation. Vol. 1: Foundations," World Scientific, 1997.

[16] Tuosto, E., "Non-Functional Aspects of Wide Area Network Programming," Ph.D. thesis, Dipartimento di Informatica, Università di Pisa, Via Buonarroti 2, 56127 Pisa - Italy (2003).

[17] Winskel, G., *Synchronization trees*, Theoretical Computer Science (1985).

A Hypergraphs Rewriting via SHR

Graph semantics is based on productions to specify edge replacement, while inference rules essentially synchronize productions and confer dynamic behaviour to graphs. We report here only the essential formal definitions; for a deeper presentation, we refer the reader to [16,15,10,8].

A transition is a logical judgment

$$\Gamma_1 \vdash G_1 \xrightarrow[\pi]{\Lambda} \gg \Gamma_2 \vdash G_2 \tag{A.1}$$

where Λ, π, Γ_2 and G_2 obey the same conditions imposed on productions. Essentially, transitions can be seen as productions having general graphs on their lhs. Hence transitions describe the dynamic evolutions of graphs.

Transition (A.1) states that $\Gamma_1 \vdash G_1$ rewrites according to constraints Λ and determines fusion substitution π. Productions are synchronized via the inference rules in Table A.1. We call *idle* a prodution having an empty set of conditions and the identity function as fusion substitution.

Definition A.1 [Graph transitions] Let $\langle \Gamma_0 \vdash G_0, \mathcal{P} \rangle$ be a graph rewriting system. The set of *transitions* $T(\mathcal{P})$ is the smallest set that contains $\mathcal{P}$, any idle production and that is closed under the inference rules in Table A.1.

A derivation is obtained by starting from the initial graph and by executing a sequence of transitions, each obtained by synchronizing productions. The synchronization of rewriting rules requires matching of the actions and unification of the third components of the constraints Λ. After productions are applied, the unification function is used to obtain the final graph by merging the corresponding nodes.

In Table A.1 we use notation $[^{v_1,\ldots,v_n}/_{u_1,\ldots,u_n}]$ (abbreviated as $[^{\boldsymbol{v}}/_{\boldsymbol{u}}]$) to denote substitutions that are applied both to graphs and sets of constraints. If $\rho = [^{\boldsymbol{v}}/_{\boldsymbol{u}}]$ is a substitution then ρG is the graph obtained by substituting all free occurrences of u_i with v_i in G for each $i = 1, ..., n$, while $\rho\Lambda = \{(x, a, \rho\boldsymbol{y}) : (x, a, \boldsymbol{y}) \in \Lambda\}$ where $\rho\boldsymbol{y}$ is the sequence $\rho(y_1), \ldots, \rho(y_h)$ if $\boldsymbol{y} = y_1, \ldots, y_h$.

Finally, given a function $f : A \to B$ and $y \in A$, $f_{-y} : A \setminus y \to B$ is defined as $f_{-y}(x) = f(x)$, for all $x \in A \setminus y$.

The most important rules in Table A.1 are (*merge*1) and (*merge*2). They regulate how nodes can be fused. Rule (*merge*1) fuses two nodes provided that no constraint is required on one of them, whereas rule (*merge*2) handles with nodes upon which complementary actions are required. Finally, rule (*par*) states how transitions on disjoint graphs can be combined together.

Rule (*merge*1) fuses nodes x and y provided that no constraint is imposed on y, i.e. $\Lambda(y) \uparrow$. Premise $x \simeq_\pi y \Rightarrow \pi(y) \neq y$ imposes that, in case y is fused with a node x such that $x \simeq_\pi y$ (namely x and y are in

$$(merge1)\quad \frac{\Gamma, y \vdash G \xrightarrow[\pi]{\Lambda}\!\!\gg \Gamma' \vdash G' \qquad \Lambda(y)\uparrow \qquad x \simeq_\pi y \Rightarrow y \neq \pi(y) \qquad \rho = \left[{}^{\pi(x)}/_{\pi(y)}\right]}{\Gamma \vdash \left[{}^{x}/_{y}\right]G \xrightarrow[(\pi;\rho)_{-y}]{\rho\Lambda}\!\!\gg \mathrm{n}(\rho\Lambda) \cup (\pi;\rho)_{-y}(\Gamma) \vdash \rho G'}$$

$$(merge2)\quad \frac{\Gamma, y \vdash G \xrightarrow[\pi]{\Lambda \cup \{(x,a,\boldsymbol{v}),(y,\overline{a},\boldsymbol{w})\}}\!\!\gg \Gamma' \vdash G' \qquad x \simeq_\pi y \Rightarrow y \neq \pi(y) \qquad \rho = mgu\{\left[{}^{[x/y]\boldsymbol{w}}/_{[x/y]\boldsymbol{v}}\right], \left[{}^{\pi(x)}/_{\pi(y)}\right]\} \qquad \Gamma'' = \mathrm{n}(\rho\Lambda) \cup (\pi;\rho)_{-y}(\Gamma) \qquad U = \rho(\Gamma') \setminus \Gamma''}{\Gamma \vdash \left[{}^{x}/_{y}\right]G \xrightarrow[(\pi;\rho)_{-y}]{(\rho\Lambda \cup (x,\tau,\langle\rangle))}\!\!\gg \Gamma'' \vdash \nu\, U.\rho G'}$$

$$(par)\quad \frac{\Gamma_1 \vdash G_1 \xrightarrow[\pi]{\Lambda}\!\!\gg \Gamma_2 \vdash G_2 \qquad \Gamma_1' \vdash G_1' \xrightarrow[\pi']{\Lambda'}\!\!\gg \Gamma_2' \vdash G_2' \qquad (\Gamma_1 \cup \Gamma_2) \cap (\Gamma_1' \cup \Gamma_2') = \emptyset}{\Gamma_1 \cup \Gamma_1' \vdash G_1 \mid G_1' \xrightarrow[\pi \cup \pi']{\Lambda \cup \Lambda'}\!\!\gg \Gamma_2 \cup \Gamma_2' \vdash G_2 \mid G_2'}$$

Table A.1
Inference rules for graph synchronization

the same equivalence class) then y must not be the representative element. However, if $x \not\simeq_\pi y$ fusion $[{}^{x}/_{y}]$ is possible; indeed, condition $x \simeq_\pi y \Rightarrow \pi(y) \neq y$ trivially holds.
A transition from $\Gamma, y \vdash G$ may be re-formulated to obtain the transition where y and x are coalesced, provided that fusion of their representative elements, ρ, is reflected on Λ, on π and on continuation $\Gamma' \vdash G'$. Indeed, if y is fused with x, also the other nodes equivalent to them are fused; the fusion substitution in the conclusion of ($merge1$) is $\pi;\rho$ (restricted to Γ), all occurrences of $\pi(y)$ are replaced with $\pi(x)$ in $\mathrm{n}(\Lambda)$ and the final graph is $\rho G'$. It is obtained by merging $\pi(y)$ and $\pi(x)$ in G'.

Rule ($merge2$) synchronizes complementary actions. The rule permits merging x and y in a transition if they offer complementary actions. As for ($merge1$), x cannot replace the representative element of its equivalence class. Most general unifier ρ takes into account possible equalities due to the transitive closure of substitutions $[{}^{\boldsymbol{v}}/_{\boldsymbol{u}}]$ after $[{}^{x}/_{y}]$ has been applied substitution. ρ fuses the corresponding nodes of the constraints and propagates previous fusions π. The resulting constraints $\rho\Lambda \cup \{(x,\tau,\langle\rangle)\}$ does not change constraints offered on nodes different from x and y (up to the necessary fusion ρ). Fusion substitution $(\pi;\rho)_{-y}$ acts on Γ by applying ρ. Finally, nodes U are those nodes that are neither in $(\pi;\rho)_{-y}(\Gamma)$ or generated by $\rho\Lambda$ are restricted in $\rho G'$; this corresponds to the *close* rule of π-calculus.

Observation A.1 *We remark that node x mentioned in rules ($merge1$) and ($merge2$) is a node in Γ appearing in the rules. This immediately follows for the condition $x \simeq_\pi y \Rightarrow y \neq \pi(y)$.*

Rule (par) simply combines together disjoint judgments. Condition $(\Gamma_1 \cup \Gamma_2) \cap (\Gamma_1' \cup \Gamma_2') = \emptyset$; guarantees that $\pi \cup \pi'$ is well defined because it implies $\Gamma_1 \cap \Gamma_1' = \emptyset$, moreover such condition states that

- graphs $\Gamma_1 \vdash G_1$ and $\Gamma_2 \vdash G_2$ are disjoint;
- names generated by the transition of $\Gamma_i \vdash G_i$ (for $i \in \{1,2\}$) do not occur neither as names of the other graph nor as names generated by its transition.

Function $\pi \cup \pi'$ applied to a node x is $\pi(x)$ or $\pi'(x)$ depending on $x \in \Gamma'$. Note that $\pi \cup \pi'$ is well defined because $\Gamma \cap \Gamma' = \emptyset$.