SHReQ: A Framework for Coordinating Application Level QoS

Dan Hirsch and Emilio Tuosto

Dipartimento di Informatica, Università di Pisa,
Largo Bruno Pontecorvo 3, I-56127, Pisa, Italy
{dhirsch,etuosto}@di.unipi.it

Abstract. We present SHReQ, a formal framework for specifying systems that handles abstract high-level QoS aspects which are becoming more and more important for *service oriented computing*. SHReQ combines *Synchronised Hyperedge Replacement* (SHR) with *constraint-semirings*. SHR is a (hyper)graph rewriting mechanism for modelling mobility and reconfiguration of systems. The novelty of the approach relies on the fact that constraint-semirings provide both the mathematics for multi-criteria QoS and the synchronisation policies underlying the SHR mechanism.

1 Introduction

Modern distributed inter-networking systems are very complex and constituted by a varied flora of architectures and communicating infrastructures. Such systems are heterogeneous, geographically distributed and highly dynamic since the communication topology can vary and the components can, at any moment, connect to or detach from the system. These features are reflected also on applications, which can be thought of as built by connecting (remote) *services*.

In a very broad sense, *Service Oriented Computing* (SOC) has been proposed as an evolutionary paradigm to build wide area distributed systems and applications. Services can be dynamically composed to provide new services, and their interactions are governed in accordance with programmable coordination policies. Web services and GRID services may be regarded as SOC and they are receiving particular attention both from academia and industry. Applications are intended as being services that search and bind to other services. In this respect, they offer a standard layer for representing data and for abstracting from the communication protocols of Internet. An ambitious goal is the automatization of the search-bind cycle so that applications can dynamically chose the "best" service available during the computation. Since the programmer does not completely control the services that her application invokes, it would be reasonable to allow her to use declarative mechanisms for expressing the "minimal" requirements on the execution environment.

Recently, *awareness of Quality of Service (QoS)* is emerging as a new exigency in both design and implementation of SOC applications. For final users, the perceived QoS of applications is not only a matter of low-level performance but also depends on application dependent requirements. For instance, the price of a given service, or the

payment mode, or else the availability of a resource (e.g., a file) in a given format are typical examples of QoS aspects that one should be able to control and/or program. Deploying distributed applications that allow programming and controlling such features is becoming more and more important and challenging. The ability of formally specifying and programming QoS aspects may represent a significant *added-value* of the SOC paradigm. Moreover, QoS information can be used to drive the design and development of application program interfaces and languages for QoS-aware middleware as well as to drive the search-bind cycle of SOC.

SOC can be naturally modelled by means of graph-based techniques, where edges represent components and nodes model the communication infrastructure. Edges sharing a node correspond to components that may interact. Systems are modelled as graphs and computations correspond to graph-rewriting. Among other proposals, *hypergraphs* and *synchronised hyperedge replacement* (SHR, for short) have been proposed for modelling distributed systems [3, 5] as a natural declarative framework. In [9, 13], SHR has been extended with mobility for modelling both architectural and programming aspects of mobile distributed applications. SHR with mobility follows a self-organising approach given by the combination of hyperedge rewriting systems for local component specification and constraint solving for coordination. Various facets of SHR have been studied with respect to issues related to distributed systems [9, 13, 10, 4, 6]. For each edge L of the system, the programmer declares its behaviour by specifying a set of productions which imposes requirements to the attachment nodes of L in order to replace L with a new hypergraph. Synchronising requirements (according to a given synchronisation policy) is the coordination mechanism allowing the evolution of systems. All SHR frameworks proposed so far do not consider quantitative aspects related to computations. In [4], SHR has been used as model for a middleware expressing quantitative aspects of applications without affecting the original synchronisation mechanism.

The main contribution of this work is SHReQ, a SHR framework for handling abstract high-level QoS aspects expressed as *constraint-semirings* [1] (c-semirings, for short). The distinguishing ingredients of SHReQ lay on embedding c-semirings in the SHR synchronisation mechanism which guides dynamic coordination/reconfiguration of systems. Namely, interactions among components (e.g., geographically distributed services) are ruled by synchronising them on events that are c-semirings values. In [4], c-semirings have been proposed as a mathematical abstraction for application-level QoS since their algebraic properties can naturally describe QoS values and the usual operations on them. For instance, multi-criteria QoS can easily be dealt with cartesian product of c-semirings, which are c-semirings as well. In [12] SHR with mobility has been generalised by parameterising the rewriting mechanism with respect to a *synchronisation algebra with mobility*. Since it is possible to turn c-semirings into synchronisation algebras, SHReQ builds on SHR where c-semirings yields the synchronisation mechanism as well as the mathematical machinery for handling QoS values.

Structure of the paper: Section 2 specifies the running example of the paper. Section 3 reports the formal definition of hypergraphs. Section 4 introduces SHReQ productions over weighted hypergraphs and formalises the running example accordingly. Section 5 defines the rewriting mechanism of SHReQ. Section 6 describes how the

rewriting mechanism applies to the running example. Section 7 concludes the paper with comments on related work, final remarks and future work.

2 A Case Study: Remote Medical Care System

This section presents our running example based on the case study of a telemedicine project carried out by Parco Scientifico e Tecnologico d'Abruzzo and University of L'Aquila detailed in [11]. The Teleservices and Remote Medical Care System (TRMCS) aims at enforcing a current trend in healthcare that is to transfer patients from hospital care to home care as quickly as feasible. TRMCS is intended to provide and guarantee assistance to at-home or mobile users. These patients do not need continuous assistance but may need prioritized assistance when urgencies happen, in which case they call a help center.

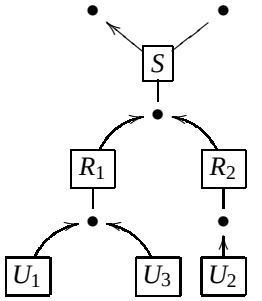


Fig. 1. A system instance

For clarity, the operations of the different components have been simplified. The system follows a hierarchical style with one server S, a variable number of routers connected to S and a variable number of users connected to the router (any user is connected to one router only). Figure 1 shows a system instance with two routers and three users (the graphical notation will be clearified later). When R_i detects an alarm from one of the connected users, it forwards the alarm requests upward to S. Server S receives alarms from R_i and it dispatches the assistance to the requesting user.

3 Syntax of Graphs

Given a set of labels $\mathcal{L}$ ranged over by L and a set of nodes $\mathcal{N}$, a *hyperedge* $L(x_1,...,x_n)$ connects nodes $x_1,\ldots,x_n \in \mathcal{N}$, where L has *rank* n (written $L:n$). We say that $x_1,...,x_n$ are the *attachment nodes* of $L(x_1,...,x_n)$. *Hypergraphs* are built from ranked hyperedges in $\mathcal{L}$ and nodes in $\mathcal{N}$.

Definition 3.1 (Hypergraphs). *A* hypergraph *is a term of the following grammar*

$$G ::= nil \mid L(x) \mid G \mid G \mid \nu\, y.G,$$

where $L:|x|$ *is a hyperedge (*$|x|$ *is the length of vector x) and* $y \in \mathcal{N}$.

Hereafter, we call hypergraphs (hyperedges) simply graphs (edges) and write $L(x)$ with the implicit assumption that $L:|x|$. Grammar in Definition 3.1 permits generating the empty graph (denoted by *nil*), graphs with a single edge, graphs built by the parallel composition of graphs and graphs where some nodes are hidden. As usual, in $\nu\, y.G$, the occurrences of y in G are bound and y is said *restricted* in G. We use $\mathrm{fn}(G)$ to denote the set of the nodes of G not occurring in the scope of a ν operator.

Example 3.1. Figure 2(a) represents the hyperedge $L(a,b,c)$ where wires connecting nodes a, b and c to L are called tentacles. The arrowed tentacle individuates the first attachment node. Moving clockwise determines the other tentacles. Figure 2(b) depicts graph $G = \nu\, z.(L(y,x,z) \mid M(x,z))$, where filled and empty circles represent free and restricted nodes, respectively.

(a) A hyperedge (b) A hypergraph

Fig. 2. Hypergraphs

Structural congruence over graphs terms allows to avoid cumbersome parenthesis.

Definition 3.2 (Structural Congruence). *The* structural congruence *is the smallest binary relation $\equiv$ over graph terms that obeys the following axioms:*

$$(G_1 \mid G_2) \mid G_3 \equiv G_1 \mid (G_2 \mid G_3) \quad G_1 \mid G_2 \equiv G_2 \mid G_1 \quad G \mid nil \equiv G$$

$$\nu\, x.\nu\, y.G \equiv \nu\, y.\nu\, x.G \quad y \notin \mathrm{fn}(G) \implies \nu\, y.G \equiv G \wedge \nu\, x.G \equiv \nu\, y.G\{y/x\}$$

$$\nu\, x.(G_1 \mid G_2) \equiv (\nu\, x.G_1) \mid G_2, \mathit{if}\ x \notin \mathrm{fn}(G_2)$$

The first row defines associativity, commutativity and identity (*nil*) for operation $|$. Axioms in the second row state that ν is a binder, i.e., the nodes of a graph can be α-renamed, restricted in any order (hence, we shorten $\nu\, x_1.\nu\, x_2 \ldots \nu\, x_n$ with $\nu\, x_1, \ldots, x_n$) and that restriction does not play any role on non-free nodes of a graph. The last axiom tunes the interplay between hiding and the parallel composition operator.

4 Graphs and productions for SHReQ

This section introduces SHReQ, a calculus based on SHR where c-semiring values are embedded in the rewriting mechanism. SHReQ takes advantage of the ideas in [4, 12], indeed, it exploits hypergraphs and SHR with mobility for modelling systems (as in [9, 13]) and c-semirings as synchronisation algebras. Hence, the rewriting mechanism of SHReQ is parameterised with respect to a given c-semiring. Basically, values of c-semirings are synchronisation actions so that synchronising corresponds to operate on c-semiring values. Moreover, a hypergraph modelling a system is decorated with c-semiring values on its nodes in order to record quantitative information on the computation of the system. A formal connection can be traced between synchronisation

algebras in the sense of [12] and c-semirings, however, page limits compel us to prove this connection in a future work.

4.1 Weighted graphs

In this context c-semirings express the requirements that a component imposes to its neighbour components and the quantitative information on computations that the environment must guarantee. They have two distinguished features that result very useful in our context. First, the cartesian product of c-semirings is still a c-semiring, hence we can uniformly deal with different types of quantities. Second, the operations of c-semirings provide a partial order on the values and a mechanism of choice. These features make c-semirings suitable for reasoning on multi-criteria QoS issues [4].

Definition 4.1 (C-semiring [1]). *An algebraic structure* $\langle S, +, \cdot, \mathbf{0}, \mathbf{1}\rangle$ *is a* constraint semiring *if* S *is a set (*$\mathbf{0}, \mathbf{1} \in S$*), and* $+$ *and* $\cdot$ *are binary operations on* S *satisfying the following properties:*

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

C-semirings are equipped with two binary operations (an additive and a multiplicative operation). The additive operation of a c-semiring induces a partial order on S defined as $a \leq_S b \iff \exists c : a + c = b$. The minimal element is thus $\mathbf{0}$ and the maximal $\mathbf{1}$.

The following examples give an intuition of some c-semiring structures.

Example 4.1. An example of c-semiring is *priority c-semiring* $P = \langle N, max, min, 0, \infty\rangle$ defined on N, the set of natural numbers with infinity. The additive operation of P is *max* (which induces the obvious order) and the multiplicative operation is *min*.

Example 4.2. Given a set of actions A, the set of *co-actions* is $\overline{A} = \{\overline{a} \mid a \in A\}$ and we let $W = A \cup \overline{A} \cup \{\mathbf{1}_W, \mathbf{0}_W, \bot\}$. The *broadcast c-semiring on* W is $\langle W, +_W, \cdot_W, \mathbf{0}_W, \mathbf{1}_W\rangle$ specified as:

$$\forall a \in Act.\, a \cdot \overline{a} = \overline{a} \wedge a \cdot a = a \tag{1}$$

$$\forall a, b \in Act \cup \overline{Act} \cup \{\bot\} : b \notin \{a, \overline{a}\} \implies a \cdot b = \bot \tag{2}$$

$$\text{the corresponding commutative rules plus the ones for } \mathbf{0} \text{ and } \mathbf{1}. \tag{3}$$

The operation $+_W$ is obtained by extending the c-semiring axioms for the additive operation with $a +_W a = a$, for all $a \in W$ and $a + b = \bot$, $\forall a, b \in Act \cup \overline{Act} \cup \{\bot\}.b \neq a$.

Hereafter, we assume a fixed c-semiring $\langle S, +, \cdot, \mathbf{0}, \mathbf{1}\rangle$.

Definition 4.2 (Weighted graphs). *A* weighted graph *is a pair* $\Gamma \vdash G$ *of a graph* G *and a* weighting function Γ *mapping a finite set of nodes to* S *such that* $\mathrm{fn}(G) \subseteq \mathrm{dom}(\Gamma)$.

A weighted graph is a graph having values in S associated to its free nodes. We write $x_1 : s_1, \ldots, x_n : s_n \vdash G$ for the weighted graph whose weighting function maps x_i to s_i, for any $i \in \{1, \ldots, n\}$, with the implicit assumptions that nodes x_i are all distinct and $\mathrm{fn}(G) \subseteq \{x_1, \ldots, x_n\}$. If $x \notin \mathrm{dom}(\Gamma)$, function $\Gamma, x : s$ is the updating of Γ on x.

In the following, we sometimes use vectors and denote the i-th component of a x by x_i, moreover, $\{|x|\} \stackrel{\text{def}}{=} \bigcup_{i \in \{1,\ldots,|x|\}} \{x_i\}$.

4.2 Productions for weighted graphs

The classical SHR approach is a declarative framework where the behaviour of an edge is specified via a set of *productions* describing the graph to be replaced in place of the edge, *provided that* some requirements are satisfied by the surrounding environment. A production takes the form $p : L(x) \xrightarrow{\Lambda} G$ where $L(x)$ is a hyperedge, G a hypergraph and Λ specifies the *requirements*. Roughly, p states that, in a given graph, an edge labelled L can be replaced by G provided that the environment satisfies requirements Λ.

Productions of SHReQ have a slightly different definition and interpretation.

Definition 4.3 (Productions). *Let* $\mathcal{R} = S \times \mathcal{N}^*$ *be the set of* requirements. *A* production *is a 4-tuple* $\chi \triangleright L(x) \xrightarrow{\Lambda} G$ *where*

- *x is a tuple of pairwise distinguished nodes and L an edge label of arity* $|x|$*;*
- $\chi : \{|x|\} \to S$ *is the* applicability function*;*
- $\Lambda : \{|x|\} \to \mathcal{R}$ *is the* communication function *assigning requirements to nodes such that for* $i \in \{1,\ldots,|x|\}$, $\Lambda(x_i) = (s,y)$ *and* $s \in Sync \implies y = \langle\rangle$. *The* communicated nodes of Λ*, denoted by* $\mathrm{n}(\Lambda)$*, are those nodes that appear in a requirement in the range of* Λ. *The set of* new nodes of Λ *is* $\mathrm{new}(\Lambda) = \mathrm{n}(\Lambda) \setminus \mathrm{dom}(\Lambda)$.
- G *is a graph such that* $\mathrm{fn}(G) \subseteq \{|x|\} \cup \mathrm{n}(\Lambda)$.

A SHReQ production, or simply production, $\chi \triangleright L(x) \xrightarrow{\Lambda} G$ states that, in order to replace L with G in a graph H, the graph H must satisfy the conditions expressed by the applicability function χ on the attachment nodes of L. Once χ is satisfied in H, L "contributes" to the rewriting by offering Λ in the synchronisation with the other edges connected to its attachment nodes. As will be more clear later, χ expressed the *minimal* requirement that the execution environment must satisfy in order to apply the production.

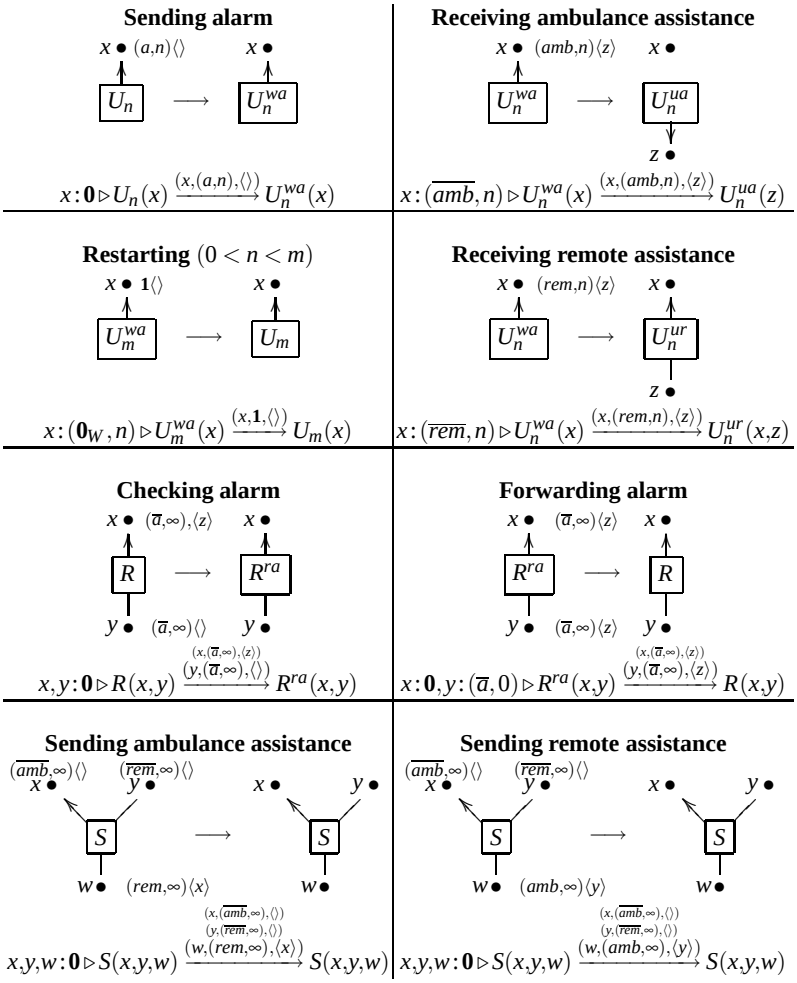
We remark that, in $\chi \triangleright L(x) \xrightarrow{\Lambda} G$, c-semiring values play different roles in χ and Λ: in the former, they are interpreted as the minimal requirements that the environment must satisfy for applying the production; in Λ they are the "contribute" that L yields to the synchronisation with the surrounding edges.

4.3 C-semirings and productions for TRMCS

The TRMCS case study can be modelled by defining productions on the c-semiring given by the cartesian product of P and B, i.e., the priority and broadcast c-semirings (Example 4.1 and 4.2). The former represents priorities among users where 1 corresponds to the highest priority and, for simplicity, all users have different priorities (i.e., $n \neq m \implies U_n \neq U_m$, this simply means that users are distinguished by their priority). The multiplication of P (i.e. *min*) chooses the user with the highest priority. The c-semiring B defined on $Act = \{rem, amb\}$ models the communication policy of TRMCS, namely broadcast synchronisation. In our example, we have the actions *rem* and *amb* (and the corresponding coactions) for alarms requesting remote or ambulance assistance, respectively.

Finally, the cartesian product of B and P yields the c-semiring, say BP, of weak broadcast together with selection of the highest priority. When clear from the context, $\mathbf{1}$ (resp. $\mathbf{0}$) denotes $\mathbf{1}_{BP}$ (resp. $\mathbf{0}_{BP}$), also $\mathbf{1} = (\mathbf{1}_W, \infty)$ and $\mathbf{0} = (\mathbf{0}_W, 0)$.

The productions for TRMCS rely on *BP* and are those collected in Figure 3. For each production, the textual and the graphical representation are given; in the latter case, drawings are simplified by not representing the applicability functions. Actually, most of them are production schemas corresponding to a set of similar productions. For example, **Sending alarm** is a production schema where a ranges over actions $\{amb, rem\}$ and n ranges over the priorities of users or else **Restarting** must be instantiated for all priorities m and n such that $0 < n < m$, hence it represents $m-1$ productions for user U_m. Moreover, we consider the following *idle* productions for users and routers:



where $a \in \{amb, rem\}$

Fig. 3. TRMCS Productions

$$x:\mathbf{0} \triangleright U_n(x) \xrightarrow{(x,\mathbf{1},\langle\rangle)} U_n(x) \qquad\qquad x:\mathbf{0},y:\mathbf{0} \triangleright R(x,y) \xrightarrow{(x,\mathbf{1},\langle\rangle),(y,\mathbf{1},\langle\rangle)} R(x,y)$$

expressing that both users and routers can remain idle during a transition without influencing the synchronisation of the other components. Hereafter, instead of writing $x:s,y:s$ as done in the idle production for R, we write $x,y:s$ whenever possible.

Intuitively, productions in Figure 3 model a scenario where router R checks for alarms and selects the user U_n with highest priority among those that have sent an alarm (ambulance or remote request are separately attended). Then, once U_n has been chosen, R communicates to S the alarm and the "address" of U_n where the requested assistance must be sent. Detailed comments on the productions follow.

Sending alarm: U_n sends an alarm (for ambulance or remote assistance) together with its priority to the router attached to node x and changes to state *waiting for assistance* (wa).

Receiving ambulance assistance: if U_n is the highest priority *wa* user then U_n disconnects from x and connects to S on node z. Notice that the applicability function requires $(\overline{amb},n)$ on attachment node x. Finally, U_n changes to *under ambulance assistance* (ua) state.

Restarting: all the other *wa* users without the highest priority, return to the initial state. The **restarting** schema applies when $(\mathbf{0},n)$ is the weight of the attachment node of U_m (for $0 < n < m$) so that user U_m silently returns to the initial state.

Receiving remote assistance: U_n moves from state *wa* to *under remote assistance* (ur) by synchronising again with R on x (analogously to **receiving ambulance assistance**). Remote assistance is modelled by making U_n and S sharing z, i.e. the node where S connects to provide assistance to U_n.

Checking alarm: this production schema states that a router R checks for alarms from its users and changes to state *responding to alarm* (ra). Indeed, R synchronises with the users connected to node y with the action $(\overline{a},\infty)$ (recall that ∞ is the $\mathbf{1}$ of P). This synchronisation yields a value $(\overline{a},n)$ recording the type of alarm to be attended (*rem* or *amb*) and the highest priority user sending the alarm.

Forwarding alarm: this production schema gets node z from user U_n that must be attended and forwards it to S (from node y to node x). Then S will share node z with the U_n. Requirements on nodes indicate the type of alarm to attend and ignores priorities given that only U_n can provide node z (the others return to the initial state). Indeed, notice that ∞ is the neutral element for the product of P.

Sending remote assistance: S checks on node w for remote alarms forwarded by a router R and connects the corresponding user to node y. This is achieved by fusing nodes y and z (provided by R using **forwarding alarm**). According to c-semiring B, server S attends one router at a time (only one router can synchronise with S). Actions on nodes x and y are used to synchronise with *ua* users.

Sending ambulance assistance: this is similar to the previous production but in this case the user is connected to node x for ambulance assistance.

Due to space limitations we do not include productions for a user after the assistance is finished, but they can be specified as done above.

5 Synchronised Rewriting for SHReQ

SHReQ rewriting mechanism relies on c-semirings where addition structure is defined. More precisely, we require that

- there are two sets *Sync* and *Fin* such that $Sync \subseteq Fin \subseteq S$ and $\mathbf{1} \in Sync$;
- there is a set $NoSync \subseteq S \setminus Fin$ such that $\forall s \in S : \forall t \in NoSync : s \cdot t \in NoSync$ and $\mathbf{0} \in NoSync$.

The intuition is that *Fin* contains those values of S representing events of complete synchronisations. This is a technical expedient from synchronisation algebras with mobility [12] for dealing with restricted nodes. Basically, values in *Fin* are those events appearing on restricted nodes that represent a "finished" synchronisation, i.e., an internal synchronisation that does not require any further interaction with the environment. A typical example might be synchronisation actions of process calculi. Among the actions in *Fin* we can select a subset of "pure" synchronisation actions, namely complete synchronisations that do not expose nodes. Set *NoSync*, on the contrary, contains the values that represent "impossible" synchronisations. As more clear later, values in *NoSync* avoid synchronisations.

Hereafter, we let Ω be a finite multiset[1] over $\mathcal{N} \times \mathcal{R}$ and, before giving SHReQ semantics, we establish some notational conventions:

- $\mathrm{dom}(\Omega) = \{x \in \mathcal{N} \mid \exists s \in S, y \in \mathcal{N}^* : (x,s,y) \in \Omega\}$;
- $\mathrm{n}(\Omega) = \bigcup_{(x,v,y)\in\Omega} \{|y|\}$, $\quad \mathrm{new}(\Omega) = \mathrm{n}(\Omega) \setminus \mathrm{dom}(\Omega)$;
- $\Omega@x = [(x,s,u) \mid (x,s,u) \in \Omega]$;
- $w_\Omega : \mathrm{dom}(\Omega) \to S$, $\quad w_\Omega : x \mapsto \prod_{(x,s,y)\in\Omega@x} s$
- given $\sigma : \mathcal{N} \to \mathcal{N}$, $\Omega\sigma = [(\sigma(x), s, u\sigma) \mid (x,s,u) \in \Omega]$.

SHReQ semantics exploits a most general unifier accounting for node fusions. We write $\mathrm{mgu}(\Omega)$ for denoting the function that yields an idempotent substitution defined if, and only if, for all $(x,s,u),(x,s',v) \in \Omega@x \setminus [(x,t,\langle\rangle) \mid t \in Sync]$

$$|u| = |v| \tag{4}$$

$$\forall i \in \{1,\ldots,|u|\} : u_i \in \mathrm{new}(\Omega) \vee v_i \in \mathrm{new}(\Omega) \tag{5}$$

$$\forall x \in \mathrm{dom}(\Omega) : card(\Omega@x) > 1 \implies w_\Omega(x) \notin NoSync \tag{6}$$

Condition (4) requires that the lengths of communicated vectors are equal. Condition (5) states that the unification cannot fuse two "old" nodes (the more general notion of [6,

[1] We write multisets by listing the (occurrences of their) elements in square brackets, e.g. $[a,a,b]$ is the multiset with two occurrences of a and one of b where the order is not important, i.e., $[a,a,b] = [a,b,a] = [b,a,a]$. Multiset membership and difference are expressed by overloading $\in$ and $\setminus$, respectively; the context will always clarify if we are referring to sets or multisets. Multiset union is denoted by $\uplus$; sometimes we also write $A \uplus B$ where either A or B are sets to denote the multiset $[a \mid a \in A] \uplus [b \mid b \in B]$.

13] can be easily re-casted in our framework). Finally, condition (6) avoids synchronisations (and hence rewritings) when a value in *NoSync* is the result of the composition. In the following, it is implicitly assumed that $\mathrm{mgu}(\Omega)$ is defined when writing $\rho = \mathrm{mgu}(\Omega)$ and that ρ is obtained by computing the most general unifier of the equations $\{u_i = v_i \mid \exists s,t \in S : (x,s,u),(x,t,v) \in \Omega \wedge 1 \leq i \leq |u|\}$.

The semantics of SHReQ is a labelled transition system specified with inference rules given on top of *quasi-productions*.

Definition 5.1 (Quasi-productions). *The set* $\mathcal{Q}_{\mathcal{P}}$ *of* quasi-productions on $\mathcal{P}$ *is defined as the smallest set containing* $\mathcal{P}$ *such that*

$$\chi \triangleright L(x) \xrightarrow{\Omega} G \in \mathcal{Q}_{\mathcal{P}} \quad \wedge \quad y \in \mathcal{N} \setminus \mathrm{new}(\Omega) \implies \chi' \triangleright L(x\{^{y}/_{x}\}) \xrightarrow{\Omega\{^{y}/_{x}\}} G\{^{y}/_{x}\} \in \mathcal{Q}_{\mathcal{P}},$$

where $x \in \{|x|\}$ *and* $\chi' : \{|x|\} \setminus \{x\} \cup \{y\} \to S$ *defined as*

$$\chi'(z) = \begin{cases} \chi(z), & \text{if } z \in \{|x|\} \setminus \{x,y\} \\ \chi(x) + s, & \text{if } z = y \wedge (y \in \{|x|\} \implies s = \chi(y)) \vee (y \notin \{|x|\} \implies s = \mathbf{0}). \end{cases}$$

Intuitively, quasi-productions are obtained by substituting nodes in productions and relaxing the condition that attachment nodes of the left-hand-side should be all different. When y is substituted for x, χ' assigns to y either $\chi(x)+\chi(y)$ or $\chi(x)$ depending whether $y \in \{|x|\}$; nodes z not involved in the substitution maintain their constraint $\chi(z)$.

Proposition 5.1. *If* $\chi \triangleright L(x) \xrightarrow{\Omega} G \in \mathcal{Q}_{\mathcal{P}}$ *then* $\mathrm{dom}(\Omega) = \{|x|\}$.

Definition 5.2 (Induced communication and weighting functions). *Let* $\mathrm{mgu}(\Omega)$ *be defined, then* $\underline{\Omega} : \mathrm{dom}(\Omega) \to \mathcal{R}$ *is the* communication function induced by Ω *defined as*

$$\underline{\Omega}(x) = \begin{cases} (\mathcal{W}_{\Omega}(x), y\rho), & \text{if } (x,s,y) \in \Omega \wedge \mathcal{W}_{\Omega}(x) \notin \mathit{Sync} \\ (\mathcal{W}_{\Omega}(x), \langle\rangle), & \text{if } (x,s,y) \in \Omega \wedge \mathcal{W}_{\Omega}(x) \in \mathit{Sync} \end{cases}$$

Let Γ *be a weighting function such that* $\mathrm{dom}(\Omega) \subseteq \mathrm{dom}(\Gamma)$, *the* weighting function induced by Γ and Ω *is* $\int_{\Omega}^{\Gamma} : \mathrm{dom}(\Gamma) \to S$, *defined as* $\int_{\Omega}^{\Gamma} : x \mapsto \begin{cases} \mathbf{1}, & x \in \mathrm{new}(\underline{\Omega}) \\ \Gamma(x), & \mathit{card}(\Omega@x) = 1 \\ \mathcal{W}_{\Omega}(x), & \text{otherwise} \end{cases}$

For each $x \in \mathrm{dom}(\Omega)$, $\underline{\Omega}$ computes the requirements resulting from the synchronisation of requirements in $\Omega@x$. More precisely, it multiplies (according to the c-semiring product) the values and applies the substitution $\mathrm{mgu}(\Omega)$ on the communicated nodes (if the resulting values are not in *Sync*). The weighting function computes the new weights of graphs after the synchronisations induced by Ω. New nodes are assigned with $\mathbf{1}$, nodes x upon which no synchronisation took place (i.e., $\mathit{card}(\Omega@x) = 1$) maintain the old weight while those where synchronisations happen (i.e., $\mathit{card}(\Omega@x) > 1$) are weighted according to the induced communication function.

We can now define the LTS of weighted graphs.

Definition 5.3 (Graph transitions). *A* SHR with QoS (SHReQ) rewriting system *consists of a pair* $(\mathcal{Q}_{\mathcal{P}}, \Gamma \vdash G)$, *where* $\mathcal{Q}_{\mathcal{P}}$ *is a set of quasi-productions on* $\mathcal{P}$ *and* $\Gamma \vdash G$ *is the initial weighted graph. The set of* transitions of $(\mathcal{Q}_{\mathcal{P}}, \Gamma \vdash G)$ *is the smallest set obtained by applying the inference rules in Table 1 where, in the rules (*REN*) and (*COM*),* $Z = \mathrm{new}(\Omega) \setminus \mathrm{new}(\underline{\Omega})$.

$$\text{(REN)}\quad \frac{\chi \triangleright L(x) \xrightarrow{\Omega} G \in \mathcal{Q}_{\mathcal{P}} \qquad \rho = \mathrm{mgu}(\Omega) \qquad \bigwedge_{x \in \mathrm{dom}(\chi)} \chi(x) \leq \Gamma(x)}{\Gamma \vdash L(x) \xRightarrow{\Omega} \int_{\Omega}^{\Gamma} \vdash (\nu\, Z)(G\rho)}$$

$$\text{(COM)}\quad \frac{\Gamma_1 \vdash G_1 \xrightarrow{\Lambda_1} \Gamma_1' \vdash G_1' \qquad \Gamma_2 \vdash G_2 \xrightarrow{\Lambda_2} \Gamma_2' \vdash G_2' \qquad \rho = \mathrm{mgu}(\Lambda_1 \uplus \Lambda_2) \qquad \bigwedge_{x \in \mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2)} \Gamma_1(x) = \Gamma_2(x)}{\Gamma_1 \cup \Gamma_2 \vdash G_1 \mid G_2 \xRightarrow{\Lambda_1 \uplus \Lambda_2} \int_{\Lambda_1 \uplus \Lambda_2}^{\Gamma_1 \cup \Gamma_2} \vdash (\nu\, Z)(G_1' \mid G_2')\rho}$$

$$\text{(RES)}\quad \frac{\Gamma, x : s \vdash G \xrightarrow{\Lambda} \Gamma', x : t \vdash G' \qquad x \notin \mathrm{n}(\Lambda) \qquad \mathfrak{s}\Lambda(x) \in \mathit{Fin}}{\Gamma \vdash (\nu\, x)G \xrightarrow{\Lambda_{/x}} \Gamma' \vdash (\nu\, x)G'}$$

$$\text{(OPEN)}\quad \frac{\Gamma, x : s \vdash G \xrightarrow{\Lambda} \Gamma', x : t \vdash G' \qquad \mathfrak{s}\Lambda(x) \in \mathit{NoSync} \cup \mathit{Fin} \qquad x \in \mathrm{n}(\Lambda)}{\Gamma \vdash (\nu\, x)G \xrightarrow{\Lambda_{/x}} \Gamma', x : t \vdash G'}$$

Table 1. Hypergraph rewriting rules.

Rule (REN) applies quasi-productions to weighted graphs provided that Ω admits a mgu and that the weights on the graphs satisfy conditions χ, namely, $\chi(x) \leq \Gamma(x)$, for all $x \in \mathrm{dom}(\chi)$. Notice that the communication function and weights in the conclusions are obtained as in Definition 5.2. Similarly, rule (COM) yields the transition obtained by synchronising the transitions of two subgraphs, provided that the (proofs of the) subgraphs assume the same weights on the common nodes. According to rule (RES) a node x can be restricted when x is not communicated and the requirements on x are in *Fin*. Namely, in order to derive a transition from $\Gamma \vdash (\nu\, x)G$, we must find a transition from a graph where x is free and the synchronisation on x has been completed. We remark that (RES) is a rule schema that must be instantiated for any $s,t \in S$. Rule (OPEN) handles the communication of restricted nodes. When a restricted node x appears in $\mathrm{n}(\Lambda)$, it can be opened provided that a transition can be found from the graph $\Gamma, x : s \vdash G$ (where x is a free node), such that either the requirement on x is a complete synchronisation or it is in *NoSync*. Rules (REN) and (COM) restrict x again when it is fused with other nodes. The latter condition allows an edge connected to x to checks whether other edges share x. Indeed, such a transition is possible only if one can compute $\mathrm{mgu}(\Lambda)$, which admits weights in *NoSync* only if the cardinality of $\Lambda@x$ is 1, namely there is only one edge connected to x. Even if this feature is not used in this paper, we remark that this is a very expressive compared to other SHR approaches.

6 SHReQ for TRMCS

This section gives a flavour of the SHReQ semantics (Table 1) by synchronising productions in Figure 3 over the graph of Figure 1. This simple example shows how SHReQ yields a general framework for dealing with system evolution and reconfiguration affected by multiple "dimensions" of quality.

First, we define sets $Sync_{BP}$, Fin_{BP} and $NoSync_{BP}$ as follows:

- $Sync_{BP} = \{\mathbf{1}\}$;
- $Fin_{BP} = \{\mathbf{1}\} \cup \{(\overline{a}, n) | \overline{a} \in W, n > 0\}$;
- $NoSync_{BP} = \{\mathbf{0}\} \cup \{(a,0) | a \in W\} \cup \{(\mathbf{0}_W, n) | n \in N\}$.

The only value of $Sync_{BP}$ is $\mathbf{1}$ so that all coactions continue to synchronise after the application of rule (1). Obviously, Fin_{BP} contains all coactions given that they are the result of any complete synchronisation (all $n > 0$ are valid priorities). Set $NoSync_{BP}$ contains all pairs with at least one $\mathbf{0}$ in their components.

In Figure 4 we give a graphical representation of two derivations where S responds to U_1 requesting for ambulance assistance. Instead of reporting the productions for each

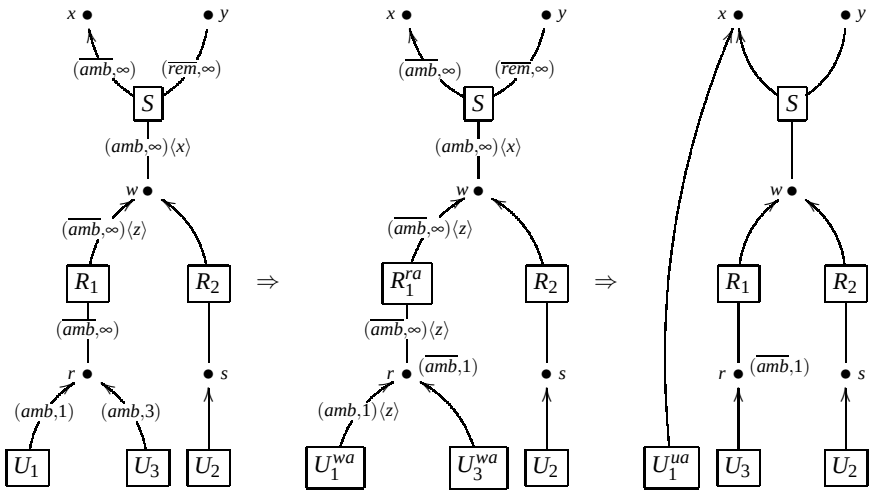


Fig. 4. A derivation for attending an ambulance alarm from U_1.

rewriting step, edge tentacles are decorated with requirements. For the sake of clarity, we index routers to refer them in a simple way, we avoid empty lists of nodes, we represent requirements $(\mathbf{1}, \langle\rangle)$ with undecorated tentacles and we report only the relevant weights with respect to the considered synchronisation.

In the first derivation, U_1 and U_3 are requesting ambulance assistance to R_1 by synchronizing (on node r) **Checking alarm** production for R_1 and **Sending alarm** for U_1 and U_3. The result of the synchronisation gives $(\overline{amb}, 1)$ as the new weight of r and U_1 as the highest priority user (second graph).

The other components do not affect this rewriting step, R_2 and U_2 apply idle productions and S applies one of its productions (which in this step produce no effect).

The second derivation produces a reconfiguration where U_1 connects to S by synchronising production **Receiving ambulance assistance** for U_1, **Forwarding alarm** for R_1 and **Sending ambulance assistance** for S (the other components apply idle productions). This is shown in the third graph of Figure 4. Moreover, the synchronisation fuses nodes z and x.

We remark that all the applicability functions of these productions are satisfied by node weights in the graphs and that productions only ensure that routers choose the highest priority user. For instance, assume that also U_2 requests assistance. It could be the case that the synchronisation among R_1, R_2 and S chooses R_2 instead of R_1, namely U_2 (instead of U_1) will connect to S. Of course, this could be resolved with productions ruling synchronisation among routers and the server in the style of those among routers and users, however, we prefer not to complicate the example with cumbersome sophistication.

Figure 5 shows part of the proof for the first rewriting step in Figure 4 corresponding to the synchronisation on node r among R_1, U_1 and U_3. The result of this sychronisation produces a broadcast synchronisation and the selection of the user with the highest priority. The first three (REN) rules are the instantiation of the three quasi-productions (we assume that the initial weight for the graph nodes is **1**). Then, the first (COM) rule (named (COM_1)) synchronises U_1 and R_1, where clearly $min(1,\infty) = 1$. Note that in the rule conclusion the weight for r contains the new value resulting from the synchronisation (i.e., $(\overline{amb},1)$). The final (COM) rule synchronises the result of (COM_1) with U_3, where $min(1,3) = 1$. We point out that in this example, all mgu are empty and in the (REN) rules the applicability conditions trivially hold by $\mathbf{0} \leq \mathbf{1}$.

7 Final Remarks

We presented SHReQ, a formal framework for specifying systems that handle abstract high-level QoS aspects which are becoming more and more important for SOC architectures. SHReQ combines SHR with c-semirings so that the former models mobility and reconfiguration of systems on top of the latter which provide both the mathematics for multi-criteria QoS and the underlying synchronisation policies.

As far as we know, SHReQ is the first to exploit an abstract definition of QoS values as a coordination mechanism. Indeed, in general QoS are either related to low-level aspects of systems or they are simply a notation for describing some non-functional properties of systems.

The work of [8] might be probably considered the closest to our proposal (from a graph transformation standpoint). This approach gives a conceptual model where structural aspects are described in a UML-like meta-model and graph transformation is used for dynamic evolution of systems. Among others, the meta-model provides a *QoS package* that influences the graph transformation rules which are required to respect the QoS package. Despite the similarities, our approach differs from [8] in the fact that in SHReQ the QoS values *are* the rewriting mechanism, not only an additional attribute.

$$\dfrac{r:\mathbf{0}\triangleright U_1(r)\xrightarrow{(r,(amb,1),\langle\rangle)}U_1^{wa}(r)}{r:\mathbf{1}\vdash U_1(r)\xrightarrow{(r,(amb,1),\langle\rangle)}r:\mathbf{1}\vdash U_1^{wa}(r)}\,(\text{REN}_1)$$

$$\dfrac{w,r:\mathbf{0}\triangleright R_1(w,r)\xrightarrow{(w,(\overline{amb},\infty),\langle z\rangle)}\xrightarrow{(r,(\overline{amb},\infty),\langle\rangle)}R_1^{ra}(w,r)}{w,r:\mathbf{1}\vdash R_1(w,r)\xrightarrow{(w,(\overline{amb},\infty),\langle z\rangle)}\xrightarrow{(r,(\overline{amb},\infty),\langle\rangle)}w,r,z:\mathbf{1}\vdash R_1^{ra}(w,r)}\,(\text{REN}_2)$$

$$\dfrac{r:\mathbf{0}\triangleright U_3(r)\xrightarrow{(r,(amb,3),\langle\rangle)}U_3^{wa}(r)}{r:\mathbf{1}\vdash U_3(r)\xrightarrow{(r,(amb,3),\langle\rangle)}r:\mathbf{1}\vdash U_3^{wa}(r)}\,(\text{REN}_3)$$

$$\dfrac{(\text{REN}_1)\qquad(\text{REN}_2)}{w,r:\mathbf{1}\vdash U_1(r)\mid R_1(w,r)\xrightarrow{(w,(\overline{amb},\infty),\langle z\rangle)}\xrightarrow{(r,(\overline{amb},1),\langle\rangle)}w,z:\mathbf{1},r:(\overline{amb},1)\vdash U_1^{wa}(r)\mid R_1^{ra}(w,r)}\,(\text{COM}_1)$$

$$\dfrac{(\text{COM}_1)\qquad(\text{REN}_3)}{w,r:\mathbf{1}\vdash U_1(r)\mid R_1(w,r)\mid U_3(r)\xrightarrow{(w,(\overline{amb},\infty),\langle z\rangle)}\xrightarrow{(r,(\overline{amb},1),\langle\rangle)}w,z:\mathbf{1},r:(\overline{amb},1)\vdash U_1^{wa}(r)\mid R_1^{ra}(w,r)\mid U_3^{wa}(r)}\,(\text{COM}_2)$$

Fig. 5. Partial proof for derivation in Figure 4.

In the area of software architecture, specific QoS aspects (e.g., dependability, performance) have been considered. For instance, in [2, 7] run-time monitoring of systems has been considered at the architectural level for handling dynamic self-adaptation that depends on (on-line) performance analysis. These approaches, aside from considering a single QoS aspect instead of application-level multi-criteria and parametric QoS, also apply traditional solutions (e.g., QoS managers) that conceptually differ from SHReQ which distributes the coordination of QoS issues over production specifications.

Modern SOCs usually specify different QoS parameters that depend on applications and should dynamically be integrated and handled. In this context, SHReQ can be generalised to a framework where edges sharing nodes are not supposed to synchronise over the same fixed c-semiring but (defining suitable composition operations among different c-semirings) one could uniformly combine heterogeneous QoS dimensions.

References

1. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *JACM*, 44(2):201–236, March 1997.
2. M. Castaldi, A. Di Marco, and P. Inverardi. Data driven reconfiguration for performance improvements: a model-based approach. In *Proceedings of RAMSS*, May 2004.

3. I. Castellani 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*, volume 153 of *LNCS*, pages 20–38. Springer-Verlag, 1983.
4. R. De Nicola, G. Ferrari, U. Montanari, R. Pugliese, and E. Tuosto. A formal basis for reasoning on programmable QoS. In N. Dershowitz, editor, *International Symposium on Verification – Theory and Practice – Honoring Zohar Manna's 64th Birthday*, volume 2772 of *LNCS*, pages 436 – 479. Springer, 2003.
5. P. Degano and U. Montanari. A model of distributed systems based on graph rewriting. *JACM*, 34:411–449, 1987.
6. G. Ferrari, U. Montanari, and E. Tuosto. A LTS semantics of ambients via graph synchronization with mobility. In *ICTCS*, volume 2202 of *LNCS*. Springer, 2001.
7. D. Garlan, B. Schmerl, and J. Chang. Using gauges for architecture-based monitoring and adaptation. In *Working Conference on Complex and Dynamic Systems Architecture*, 2001.
8. P. Guo and R. Heckel. Conceptual modeling of styles for mobile systems: A layered approach based on graph transformation. In *Proc.of IFIP TC8 Working Conference on Mobile Information Systems(MOBIS))*, pages 65–78. Springer Verlag, 2004.
9. D. Hirsch. *Graph Transformation Models for Software Architecture Styles*. PhD thesis, Departamento de Computación, UBA, 2003. http://www.di.unipi.it/˜dhirsch.
10. D. Hirsch, P. Inverardi, and U. Montanari. Reconfiguration of Software Architecture Styles with Name Mobility. In A. Porto and G.-C. Roman, editors, *Coordination 2000*, volume 1906 of *LNCS*, pages 148–163. Springer Verlag, 2000.
11. P. Inverardi and H. Muccini. The Teleservices and Remote Medical Care System (TRMCS). In *10th IWSSD*, San Diego, California, November 2000.
12. I. Lanese and U. Montanari. Synchronization algebras with mobility for graph transformations. In *Proc. Foundations of Global Ubiquitous Computing*, ENTCS, 2004. To appear.
13. E. Tuosto. *Non-Functional Aspects of Wide Area Network Programming*. PhD thesis, Dipartimento di Informatica, Università di Pisa, May 2003. TD-8/03.