A Basic Calculus for Modeling Service Level Agreement*

Rocco De Nicola[1], Gianluigi Ferrari[2], Ugo Montanari[2],
Rosario Pugliese[1], and Emilio Tuosto[2]

[1] Dipartimento di Sistemi e Informatica,
Università di Firenze, Via C. Lombroso 6/17, 50134 Firenze – Italy
email: {denicola,pugliese}@dsi.unifi.it
[2] Dipartimento di Informatica,
Largo Pontecorvo 1, 56127 Pisa – Italy
email: {giangi,ugo,etuosto}@di.unipi.it

Abstract. The definition of suitable abstractions and models for identifying, understanding and managing Quality of Service (QoS) constraints is a challenging issue of the Service Oriented Computing paradigm. In this paper we introduce a process calculus where QoS attributes are first class objects. We identify a minimal set of primitives that allow capturing in an abstract way the ability to control and coordinate services in presence of QoS constraints.

1 Introduction

Service Oriented Computing (SOC) [14] has been proposed as an evolutionary paradigm to build wide area distributed systems and applications. In this paradigm, services are the basic building blocks of applications. Services are heterogeneous software components which encapsulate resources and deliver functionalities. Services can be dynamically composed to provide new services, and their interaction is governed in accordance with programmable coordination policies. Examples of SOC architectures are provided by WEB services and GRID services.

The SOC paradigm has to face several challenges like service composition and adaptation, negotiation and agreement, monitoring and security. A key issue of the paradigm is that services must be delivered in accordance with *contracts* that specify both client requirements and service properties. These contracts are usually called Service Level Agreements (SLA). SLA contracts put special emphasis on Quality of Service (QoS) described as a set of non functional properties concerning issues like response time, availability, security, and so on.

The actual metric used for evaluating QoS parameters is heavily dependent on the chosen level of abstraction. For instance, when designing network infrastructures, performance (with some probabilistic guarantees) is the main QoS metric. When describing multimedia applications, visual and audible qualities would be the crucial parameters. Instead, for final users, the perceived QoS is not just a matter of performance

* Work partially supported by EU-FET Project AGILE, EU-FET Project MIKADO, EU-FET Project PROFUNDIS, and MIUR project SP4 Architetture Software ad Alta Qualità di Servizio per Global Computing su Cooperative Wide Area Networks.

but also involves availability, security, usability of the required services. Moreover, the user would like to have a certain control on QoS parameters in order to customize the invoked services, while network providers would like to have a strict control over services. The resolution of this tension will be inherently dynamic depending on the run-time context.

In our view, it is of fundamental importance to develop formal machineries to describe, compose and relate the variety of QoS parameters. Indeed, the formal treatment of QoS parameters would contribute to the goal of devising robust programming mechanisms and the corresponding reasoning techniques that naturally support the SOC paradigm. In this paper we face this issue by introducing a process calculus where QoS parameters are used to control behaviours, i.e. QoS parameters are first class objects.

The goal of the present paper is to identify a minimal set of constructs that provide an abstract model to control and coordinate services in presence of QoS constraints. This differentiates our proposal from other approaches. In particular, process calculi have been designed to model QoS in terms of performance issues (e.g., the probabilistic π-calculus [15]). Other process calculi have addressed the issues of failures and failure detection [13]. Process calculi equipped with powerful type systems have also been put forward to describe the behavioral aspects of contracts [11, 10, 9].

Some preliminary results towards the direction of this paper can be found in [3, 4]. Cardelli and Davies [3] introduced a calculus which incorporates a notion of communication rate (bandwidth) together with some programming constructs. In [17, 8, 4] a (hyper)graph model to control explicitly QoS attributes has been introduced. The graphical semantics allows us to describe interactions in accordance with the agreed QoS level as optimal paths in the model thus creating a bridge between formal models and the protocols used in practice. Here, we elaborate on [4] with the aim of bridging further the gap between formal theories and the pragmatics of software development.

Fundamental to our approach is the notion of QoS values; a QoS value is a *tuple* of values and each component of the tuple indicates a QoS dimension. The values of the fields can be of different kind, for instance, the value along the latency dimension could be a numerical value but the security values could have the form of sets of capabilities indicating the permissions to perform some operations on given resources, e.g. read or write a file. Compositionality of QoS values is therefore a key element of our approach: the composition of QoS values will be a QoS value as well. Indeed, one might want to build a QoS value based on latency, availability and access rights of a service.

To guarantee compositionality of QoS parameters, we shall require QoS values to be elements of suitable algebraic structures, called *constraint semirings* (c-semirings, for short), consisting of a domain and two operations, the *additive* (+) and the *multiplicative* ($\cdot$) operations, satisfying some properties. The basic idea is that the former is used to select among values and the latter to combine values. C-semirings were originally proposed to describe and program constraints problems [2]. Several semirings have been proposed to model QoS issues. For instance, general algorithms for computing shortest paths in a weighted directed graph are based on the structure of semirings [12]. The modelling of trustness in an ad hoc networks exploits the semiring structure [16]. C-semiring based methods have a unique advantage when problems with multiple

QoS dimensions must be tackled. In fact, it turns out that cartesian products, exponentials and power constructions of c-semirings are c-semirings as well.

Our process calculus, $\mathcal{K}oS$, builds on K (*Kernel Language for Agent Interaction and Mobility*) [5]. K is an experimental kernel programming language specifically designed to model and program wide area network applications with located services and mobility. K naturally supports a *peer-to-peer* programming model where interacting peers (nodes in K terminology) cooperate to provide common sets of services. $\mathcal{K}oS$ primitives handle QoS values as first class entities. For instance, an overlay network is specified by creating nodes ($node_\kappa\langle t\rangle$) and new links ($s \overset{\kappa}{\frown} t$) and indexing them with the QoS value κ of the operation. Thus, for instance the expression $s \overset{\kappa}{\frown} t$ states that s and t are connected by a link whose QoS parameters are given by κ.

The operational semantics of $\mathcal{K}oS$ ensures that the QoS values are respected during system evolution. Suppose for example that node s would interact by an operation whose QoS value is κ' with node t along the link $s \overset{\kappa}{\frown} t$. This interaction will be allowed provided that the SLA contract of the link is satisfied, namely, $\kappa' \leq \kappa$.

We shall illustrate the expressiveness of the calculus through several examples. This can appear as an exercise in coding a series of linguistic primitives into our calculus notation, but it yields much more because the encodings offer a practical illustration of how to give a precise semantic interpretation of QoS management. Indeed, the main contribution of this paper is the careful investigation of a minimal conceptual model which provides the basis to design programming constructs for SOC. We focus on the precise semantic definition of the calculus because it is a fundamental step to design programming primitives together with methods supporting the correct usages of the primitives and the formal verification of the resulting applications.

The rest of the paper is organized as follows. In the next section we illustrate a motivating example and, in Section 3, we introduce syntax and semantic of $\mathcal{K}oS$. In Section 4, we deal with expressivity issues and in the subsequent one we present a more complex scenario and show how it can be tackled by following our approach.

2 A motivating example

Before introducing the formal definition of $\mathcal{K}oS$, we prefer to show its usefulness by modelling a realistic, but simplified, example. Our purpose here is to give a flavour of the underlying programming paradigm. We consider a scenario where n servers provide services to m clients and we focus on balancing the load of the servers. Clients and servers are located on different nodes; a generic client node has address c_i while a generic server node has address s_j. Clients issue requests to servers by spawning process R from their node to a server node. For simplicity, we abstract from the actual structures of QoS values, and we assume that clients and servers "knows" each other and cannot be created dynamically. Adding dynamicity is straightforward.

A generic client node M_i, for $i \in \{1, \ldots, m\}$, is described by the following term:

$$M_i \stackrel{\text{def}}{=} c_i :: \langle s_1, \kappa_1\rangle \mid \ldots \mid \langle s_n, \kappa_n\rangle \mid !C_\delta.$$

Intuitively, M_i represents a network component with address c_i, containing tuples of the form $\langle s_j, \kappa_j\rangle$, for $j \in \{1, \ldots, n\}$, and running process $!C_\delta$. Each tuple $\langle s_j, \kappa_j\rangle$ represents

the load κ_j of the server s_j that the client perceives, thus the whole set of tuples represents a sort of *directory service* containing the SLA contract with the available servers. Operator ! is the replication operator: $!C_\delta$ represents an unbounded number of concurrent copies of process C_δ. Finally, process C_δ specifies the behaviour of the client and is defined as follows:

$$C_\delta \stackrel{\text{def}}{=} (?u, ?v).\varepsilon_v[R]@u.con_{v\cdot\delta}\langle u\rangle.\langle u, v\cdot\delta\rangle.$$

Initially, the client selects a server by non-deterministically inputting a tuple by means of the operation $(?u, ?v)$. Once the input is executed, variables u and v are instantiated with the server name and its load, respectively. Afterward, the client tries to spawn process R to the selected server u. Execution of $\varepsilon_v[R]@u$ takes place only if a "suitable" link toward u exists. What here is meant for "suitable" is that the load v of the client must not exceed the value on the link. Then, since remote spawning consumes the links traversed during the migration, the client attempts to re-establish a connection with u by executing $con_{v\cdot\delta}\langle u\rangle$. Notice that the operation $con_{v\cdot\delta}\langle u\rangle$ is used by the client to ask for a link with a QoS value increased of a quantity δ. Once the connection has been established, the client updates its SLA view of the servers load by inserting tuple $\langle u, v\cdot\delta\rangle$ into its local directory service.

A generic server N_j, for $j \in \{1, \ldots, n\}$, is described as follows:

$$\begin{aligned} N_j \stackrel{\text{def}}{=}\ & s_j :: \langle h\rangle \ \mid\ \langle c_1, \kappa'_1\rangle \ \mid\ \ldots \ \mid\ \langle c_m, \kappa'_m\rangle \ \mid \\ & !(S\ c_1\ s_j) \ \mid\ \ldots \ \mid\ !(S\ c_m\ s_j). \end{aligned}$$

Similarly to clients, N_j encapsulates a directory service containing SLA data about the clients. This directory service is formed by tuples of the form $\langle c_i, \kappa'_i\rangle$, for $i \in \{1, \ldots, m\}$, each recording the QoS value κ'_i assigned to the link towards node c_i, and by the current load of the server, represented by a tuple containing a natural number $\langle h\rangle$. For any client c_i there is a *load manager* $S\ c_i\ s_j$ which decides whether a link with c_i can be re-established or not. Process $S\ c\ s$ is written as follows:

$$\begin{aligned} S\ c\ s \stackrel{\text{def}}{=}\ & (?l).\langle l\rangle.If_s\ l < max \\ & \quad then\ (c, ?v).acc_{f(v,l)}\langle c\rangle.\langle c, f(v, l)\rangle. \end{aligned}$$

The load manager repeatedly acquires the tuple $\langle h\rangle$ (current load) and compares it with the maximum admissible load (max). Then, the process decides whether to accept requests for new connections coming from the client: the link is created only when h is less than max. The QoS value of the new link is computed by a function f and depends on both the old QoS value and the current load.

Finally, we assume that process R representing clients service requests is a sequential process of the form

$$R \stackrel{\text{def}}{=} (?x).\langle x+1\rangle \ldots actual\ request \ldots (?y).\langle y-1\rangle,$$

Namely, R has a prologue and an epilogue which respectively increments and decrements the counter that measures the server load.

3 The calculus

This section introduces $\mathcal{K}oS$ a calculus that provides a set of basic primitives for modelling and managing QoS values. A $\mathcal{K}oS$ term represents a net made of *nodes* which model places where computations take place or where services can be allocated/accessed. We assume as given a set of nodes $\mathcal{S}$ (ranged over by s, t, ...) that are connected by *links* representing the middleware infrastructure, i.e., the interactions between two nodes can take place only if they are connected by a sequence of links. Links are weighted by "measures" that represent the QoS value of the connections.

3.1 QoS Values as constraint semirings

We assume existence of a set of *QoS values* C, ranged over by κ, that forms a *constraint semiring* [2] (*c-semiring*).

Definition 1 (C-semiring). *An algebraic structure* $\langle A, +, \cdot, \mathbf{0}, \mathbf{1}\rangle$ *is a c-semiring if* A *is a set (*$\mathbf{0}, \mathbf{1} \in A$*), and* $+$ *and* $\cdot$ *are binary operations on* A *that satisfy the following properties:*

- $+$ *(additive operation) is commutative, associative, idempotent,* $\mathbf{0}$ *is its unit element and* $\mathbf{1}$ *is its absorbing element;*
- $\cdot$ *(multiplicative operation) is commutative, associative, distributes over* $+$, $\mathbf{1}$ *is its unit element, and* $\mathbf{0}$ *is its absorbing element.*

Operation $+$ induces a partial order on A defined as $a \leq_A b \iff a + b = b$. The minimal element is thus $\mathbf{0}$ and the maximal $\mathbf{1}$. $a \leq_A b$ means that a is more constrained than b.

An example of c-semiring is $\langle \omega, min, +, +\infty, 0\rangle$, where ω is the set of natural numbers, the minimum between natural numbers is the additive operation and the sum over natural numbers is the multiplicative operation. Notice that in this case the partial order induced by the additive operations is the inverse of the ordinary total order on natural numbers. Another example of c-semiring is $\langle \wp(\{A\}), \cup, \cap, \emptyset, A\}\rangle$, where $\wp(A)$ is the powerset of a set A, and $\cup$ and $\cap$ are the usual set union and intersection operations.

$\mathcal{K}oS$ does not take a definite standing on which of the many c-semiring structures to use. The appropriate c-semiring to work with should be chosen, from time to time, depending on the kind of QoS dimensions one intends to model. Below, we introduce some c-semiring structures together with the QoS dimension they handle:

- $\langle \{true, false\}, \vee, \wedge, false, true\rangle$ (boolean): Network and service availability.
- $\langle$Real+$, min, +, +\infty, 0\rangle$ (optimization): Price, propagation delay.
- $\langle$Real+$, max, min, 0, +\infty\rangle$ (max/min): Bandwidth.
- $\langle [0, 1], max, \cdot, 0, 1\rangle$ (probabilistic): Performance and rates.
- $\langle [0, 1], max, min, 0, 1\rangle$ (fuzzy): Performance and rates.
- $\langle 2^N, \cup, \cap, \emptyset, N\rangle$ (set-based, where N is a set): Capabilities and access rights.

C-semiring based methods have a unique advantage when problems with multiple QoS criteria must be tackled. In fact, it turns out that cartesian products, exponentials and power constructions of c-semirings are c-semirings as well.

N, M	::=		N
		$\mathbf{0}$	*Empty net*
	|	$s :: P$	*Located Process*
	|	$s \overset{\kappa}{\frown} t$	*Link*
	|	$(\nu\, s)N$	*Node restriction*
	|	$N \parallel M$	*Net composition*
P, Q	::=		P
		$\mathbf{0}$	*Null process*
	|	$\gamma.P$	*Action prefixing*
	|	$(\nu\, s)P$	*Restriction*
	|	$P \mid Q$	*Parallel process*
	|	$!P$	*Iteration*
γ	::=		P
		$node_\kappa\langle t\rangle$	*Node creation*
	|	$con_\kappa\langle t\rangle$	*Connection request*
	|	$acc_\kappa\langle t\rangle$	*Connection acceptance*
	|	(T)	*Input*
	|	$\langle v_1, \ldots, v_n\rangle$	*Output*
	|	$\varepsilon_\kappa[P]@t$	*Remote process spawning*
T	::=	$\varepsilon \mid v \mid ?x \mid \neg v \mid T, T$	I

Table 1. $\mathcal{K}oS$ Syntax

3.2 Syntax

The syntax of $\mathcal{K}oS$ is presented in Table 1. Other than the existence of C, existence of a set of *names* $\mathcal{N}$ (ranged over by r, s and t) is assumed. First-class *values*, ranged over by u and v, can be either QoS values or names.

The syntax for nets permits the (possibly empty) parallel composition of *located processes* and *links*. A located process $s :: P$ consists of a name s, called the *address* of P, and the process P running at s. A *link* $s \overset{\kappa}{\frown} t$ states that s and t are connected by a link whose QoS value is κ. The net $(\nu\, s)N$ is a net that declares s as restricted in N, which is the scope of the restriction.

The syntax for processes is standard. The symbol $\mathbf{0}$ overloads the symbol for empty nets; however, the contexts will clarify whether it refers to processes or nets. Prefixes γ encompass actions for

- creating a node ($node_\kappa\langle t\rangle$) or a connection to/from another node ($con_\kappa\langle t\rangle$, $acc_\kappa\langle t\rangle$),
- exchanging tuples of values ((T) and $\langle v_1, \ldots, v_n\rangle$),
- remotely spawning a process ($\varepsilon_\kappa[P]@t$).

Links are oriented, indeed $s \overset{\kappa}{\frown} t$ allows a process to be spawned from s to t but not the viceversa. The creation of new links is obtained by synchronising actions $con_\kappa\langle t\rangle$ and $acc_{\kappa'}\langle s\rangle$ performed at s and t, respectively.

Communication involves exchange of tuples (i.e. finite sequences) of values that are retrieved via pattern matching. Input prefixes use *templates* T, namely finite sequences

of values or placeholders (written as $?x$). Execution of an output prefix causes generation of a tuple of values $v_1, \ldots, v_n$. Both the empty template and the empty tuple are denoted by ε. Hereafter, we let $\mathfrak{t}$ range over tuples of values and, given a template T and a tuple $\mathfrak{t}$, we let T_i and $\mathfrak{t}_i$ denote the i-th element of T and $\mathfrak{t}$, respectively.

The placeholder $?x$ binds the occurrences of x in the rest of the template, namely, in $?x, T$, the scope of $?x$ is T. The set $\mathrm{bn}(T)$ collects the names bound in T while $\mathrm{fn}(T)$ denotes the names having free occurrences in T; their definitions are standard. We consider as equivalent those templates that differ only for renaming of bound names. The template $\neg v$ tests for inequality, namely, it requires the matching tuple to contain a value different from v (see Definition 7). The only binders of the calculus are the placeholder $?x$ and the node restriction $\nu\, s$. Note that node names might be QoS values (e.g., for specifying access rights), hence, we write $\mathrm{fn}(\kappa)$ to denote the names appearing in κ. Moreover, we require that QoS values do not bind node names, therefore, $\mathrm{bn}(\kappa)$ is empty, for any QoS value κ. We formally define *free* and *bound* names of nets and processes as follows. In the following we write $\mathrm{fn}(\_, \_)$ (resp. $\mathrm{bn}(\_, \_)$) as an abbreviation for $\mathrm{fn}(\_) \cup \mathrm{fn}(\_)$ ($\mathrm{bn}(\_) \cup \mathrm{bn}(\_)$, respectively).

Definition 2 (Free and bound names). *The free names of prefix actions are defined as expected:* $\mathrm{fn}(\gamma) = \mathrm{fn}(\kappa) \cup \{s\}$, *if* $\gamma \in \{node_\kappa\langle s\rangle, con_\kappa\langle s\rangle, acc_\kappa\langle s\rangle\}$, $\mathrm{fn}((T)) = \mathrm{fn}(T)$, $\mathrm{fn}(\langle v_1, \ldots, v_n\rangle) = \mathrm{fn}(v_1) \cup \ldots \cup \mathrm{fn}(v_n)$ *and* $\mathrm{fn}(\varepsilon_\kappa[P]@s) = \mathrm{fn}(\kappa, P) \cup \{s\}$. *Bound names of* γ *are defined similarly, e.g.,* $\mathrm{bn}((T)) = \mathrm{bn}(T)$ *and* $\mathrm{bn}(\varepsilon_\kappa[P]@s) = \mathrm{bn}(P)$ *(while in the remaining cases is the empty set).*

The sets $\mathrm{fn}(\_)$ *and* $\mathrm{bn}(\_)$ *of free and bound names of processes and nets are defined accordingly. The only non-standard case is that for links where we let* $\mathrm{fn}(r \overset{\kappa}{\frown} s) = \mathrm{fn}(\kappa) \cup \{s, r\}$ *and* $\mathrm{bn}(r \overset{\kappa}{\frown} s) = \emptyset$.

As usual, processes or nets obtained by α-converting bound names are considered equivalent. Moreover, we assume the following structural congruence laws.

Definition 3 (Structural congruence). *The relation* $\equiv_P \subseteq P \times P$ *is the least equivalence relation on processes (containing* α*-conversion and) satisfying the following axioms:*

- $(P, \mid, \mathbf{0})$ *is a commutative monoid;*
- $!P \equiv_P P \mid !P$.

The relation $\equiv \subseteq N \times N$ *is the least equivalence relation on nets (containing* α*-conversion and) satisfying the following axioms:*

- $(N, \parallel, \mathbf{0})$ *is a commutative monoid;*
- *if* $P \equiv_P Q$ *then* $s :: P \equiv s :: Q$;
- $s :: P \mid Q \equiv s :: P \parallel s :: Q$;
- $s :: (\nu\, t)P \equiv (\nu\, t)(s :: P)$, *if* $t \neq s$;
- $(\nu\, s)(N \parallel M) \equiv N \parallel (\nu\, s)M$, *if* $s \notin \mathrm{fn}(N)$;
- $(\nu\, s)(\nu\, t)N \equiv (\nu\, t)(\nu\, s)N$.

The last axiom of Definition 3 states that the order of the restrictions is irrelevant, hence we can write $(\nu\, s_1, \ldots, s_n)N$ instead of $(\nu\, s_1) \ldots (\nu\, s_n)N$.

3.3 Semantics

We define the operational semantics of $\mathcal{K}oS$ by means of a labelled transition system that describes the evolution of nets. In the semantic clauses, it is useful to define a function that, given a net N, yields the names that are used as node addresses in the net.

Definition 4 (Addresses). *Let* addr *be the function given by:*

$$\mathtt{addr}(N) = \begin{cases} \emptyset, & N = \mathbf{0} \vee N = s \overset{\kappa}{\frown} t \\ \{s\}, & M = s :: P \\ \mathtt{addr}(M) \setminus \{s\}, & N = (\nu\, s)M \\ \mathtt{addr}(N_1) \cup \mathtt{addr}(N_2), & N = N_1 \parallel N_2. \end{cases}$$

Notice that $\mathtt{addr}(N) \subseteq \mathrm{fn}(N)$, but not necessarily $\mathtt{addr}(N) = \mathrm{fn}(N)$, for instance if $N = s :: \langle t \rangle.\mathbf{0}$ then $\mathrm{fn}(N) = \{s, t\}$ while $\mathtt{addr}(N) = \{s\}$. Basically, $\mathtt{addr}(N)$ collects those free names of N that effectively occur in N as address of some node.

Definition 5 (Localized Actions). *Let γ be a prefix, then the* localized prefix *$\gamma@s$ is defined as follows:*

$$\gamma@s = \begin{cases} s\ \varepsilon^{s}_{\kappa}\langle P \rangle@t & \text{if } \gamma = \varepsilon_{\kappa}\langle P \rangle@t \\ s\ \gamma & \text{otherwise} \end{cases}$$

The syntax of localized actions *α is given below:*

$$\alpha ::= \gamma@s \quad | \quad s\ \mathit{link}\ t \quad | \quad \tau$$

We let $\mathrm{fn}(\gamma@s) = \mathrm{fn}(\gamma) \cup \{s\}$ and $\mathrm{bn}(\gamma@s) = \mathrm{bn}(\gamma)$.

Definition 6 (Nets semantics). *The operational semantics of nets is given by the relation $\rightarrow \subseteq N \times (\alpha \times C) \times N$. Relation $\rightarrow$ is defined by the rules in Table 2 and the following standard rules:*

$$(\quad)\ \frac{N \xrightarrow[\kappa]{\tau} M}{(\nu\, s)N \xrightarrow[\kappa]{\tau} (\nu\, s)M} \qquad (\quad)\ \frac{N \equiv N' \xrightarrow[\kappa]{\alpha} M' \equiv M}{N \xrightarrow[\kappa]{\alpha} M}$$

$$(\quad)\ \frac{N \xrightarrow[\kappa]{\alpha} N'}{N \parallel M \xrightarrow[\kappa]{\alpha} N' \parallel M}\ \text{if}\ \begin{cases} \mathrm{bn}(\alpha) \cap \mathrm{fn}(M) = \emptyset \quad \wedge \\ (\mathtt{addr}(N') \setminus \mathtt{addr}(N)) \cap \mathtt{addr}(M) = \emptyset \end{cases}$$

Intuitively, $N \xrightarrow[\kappa]{\alpha} M$ states that the net N can perform the transition α to M by exposing the QoS value κ. Clearly, all local transitions (communications, node or link creations) have unitary QoS value, while the only non-trivial QoS values appear on the transitions that spawn processes or show the presence of links. Let us give more detailed comments on the rules in Table 2.

Rule ( ) states that a link within a net disappears once it has been used. These transitions are used in the premises of rules ( ) and ( ) for establishing a path between two nodes such that a remote evaluation can take place.

Rule ( ) accounts for action prefixing; node creation, however, deserves a specific treatment that is defined in rule ( ). The side condition of ( ) also states that

( ) $s \overset{\kappa}{\frown} t \xrightarrow[\kappa]{s \text{ link } t} \mathbf{0}$

( ) $s :: \gamma.P \xrightarrow[\mathbf{1}]{\gamma@s} s :: P, \gamma \notin \{node_\kappa\langle t\rangle, con_\kappa\langle s\rangle, acc_\kappa\langle s\rangle\}$

( ) $s :: node_\kappa\langle t\rangle.P \xrightarrow[\mathbf{1}]{node\langle t\rangle} s :: P \parallel s \overset{\kappa}{\frown} t \parallel t :: \mathbf{0}, s \neq t$

( ) $$\frac{N \xrightarrow[\mathbf{1}]{s \; con_\kappa\langle t\rangle} N' \qquad M \xrightarrow[\mathbf{1}]{t \; acc_{\kappa'}\langle s\rangle} M'}{N \parallel M \xrightarrow[\mathbf{1}]{\tau} N' \parallel M' \parallel s \overset{\kappa}{\frown} t} \kappa \leq \kappa'$$

( ) $s :: \varepsilon_\kappa[Q]@s.P \xrightarrow[\mathbf{1}]{\tau} s :: P \parallel s :: Q$

( ) $$\frac{N \xrightarrow[\kappa']{r \; \varepsilon_\kappa^s\langle P\rangle@t} N' \qquad M \xrightarrow[\kappa'']{r \; link \; r'} M' \qquad \kappa' \cdot \kappa'' \leq \kappa}{N \parallel M \xrightarrow[\kappa' \cdot \kappa'']{r' \; \varepsilon_\kappa^s\langle P\rangle@t} N' \parallel M'}, t \neq r'$$

( ) $$\frac{N \xrightarrow[\kappa']{r \; \varepsilon_\kappa^s\langle P\rangle@t} N' \qquad M \xrightarrow[\kappa'']{r \; link \; t} M' \qquad \kappa' \cdot \kappa'' \leq \kappa}{N \parallel M \xrightarrow[\kappa' \cdot \kappa'']{\tau} N' \parallel M' \parallel t :: P}$$

( ) $$\frac{N \xrightarrow[\mathbf{1}]{s \; (T)} N' \qquad M \xrightarrow[\mathbf{1}]{s \; \mathfrak{t}} M' \qquad \bowtie (T, \mathfrak{t}) = \sigma}{N \parallel M \xrightarrow[\mathbf{1}]{\tau} N'\sigma \parallel M'}$$

Table 2. Network semantics

no link from s to itself can be created. Indeed, we assume that transitions that involve only the local node have unitary QoS value and are always enabled.

Rule ( ) allows a process allocated at s to use a name t as the address of a new node and to create a new link from s to t exposing the QoS value κ. The side condition of ( ) prevents that new nodes (and links) are created by using addresses of existing nodes.

Rule ( ) adds a new link between two existing addresses s and t; the link is created only if the processes at s and t satisfy the SLA contract. More precisely, the accepting node t is willing to connect only to those nodes that declare a QoS value lower than κ'. If this condition holds, a new link is added to the net, such link has the QoS value exposed by s. One can think of s as asking for the connection with *at least* some characteristics

expressed by κ and t establishes the connection only when it can enforce the requirement of s, namely $\kappa \leq \kappa'$.

Rule ( ) states that the local spawning of a process is always enabled while rules ( ) and ( ) control process migration and require more detailed explanations. A remote spawning action $\varepsilon_\kappa[P]@t$ consists of the migrating process P, the arrival node t and a QoS value κ expressing that P must be routed on a path exposing a QoS value[1] at most κ. Differently from the local spawning of processes, remote spawning is not always possible, it is indeed mandatory that the net contains a path of links from the starting node s to the arrival node t. Moreover, the SLA contract of the path between s and t must not exceed the value κ that the spawner has declared. Notice that this semantically describes the SLA agreement on the mobility of processes. This is formally achieved by rules ( ) and ( ). More specifically, rule ( ) states that, if the migrating process can go through an intermediate node r and a link from r to a node $r' \neq t$ exists, the QoS value κ' of the partial path from s to r composed with the value κ'' of the link from r to r' must be lower than κ. If this is the case, a transition can be inferred stating that P, spawned from s, can go through r' exposing the QoS value $\kappa' \cdot \kappa''$. Rule ( ) is similar to ( ) but describes the last hop of P, namely when the target node t is reached. In this case, P is spawned at t, provided that the QoS value of the whole path that has been found is lower than κ.

Rule ( ) establishes that a synchronization takes place provided that sender and receiver are allocated at the same node and that the template and the tuple match according to the definition below. Hereafter, we use σ to denote a substitution, i.e. a map from names to names and QoS values, and $\sigma[\sigma']$ to denote the composition of substitutions, i.e. the substitution σ'' defined as follows: $\sigma''(x) = \sigma'(x)$ if $x \in \mathrm{dom}(\sigma')$, $\sigma''(x) = \sigma(x)$ if $x \in \mathrm{dom}(\sigma) - \mathrm{dom}(\sigma')$.

Definition 7 (Pattern matching). *A template T and a tuple* $\mathfrak{t}$ match *when the following function is defined*

$$\bowtie(T, \mathfrak{t}) = \begin{cases} \varepsilon & \text{if } (T = \varepsilon \ \wedge \ \mathfrak{t} = \varepsilon) \vee (T = v \ \wedge \ \mathfrak{t} = v) \\ \varepsilon & \text{if } T = \neg v \ \wedge \ \mathfrak{t} = v' \ \wedge \ v \neq v' \\ \{^v/_x\} & \text{if } T = ?x \ \wedge \ \mathfrak{t} = v \\ \sigma[\sigma'] & \text{if } T = F, T' \ \wedge \ \mathfrak{t} = v, \mathfrak{t}' \wedge \ \bowtie(F, v) = \sigma \ \wedge \ \bowtie(T'\sigma, \mathfrak{t}') = \sigma' \end{cases}$$

where the application of a substitution to a template, $T\sigma$, is defined as follows:

$$T\sigma = \begin{cases} \varepsilon & \text{if } T = \varepsilon \\ v, T'\sigma & \text{if } T = x, T' \ \wedge \ \sigma(x) = v \\ x, T'\sigma & \text{if } T = x, T' \ \wedge \ x \notin \mathrm{dom}(\sigma) \\ ?x, T'\sigma\{^x/_x\} & \text{if } T = ?x, T'. \end{cases}$$

Under the conditions of ( ), the substitution $\bowtie(T, \mathfrak{t})$ is applied to the receiver. Note that $\bowtie$ may not be defined, for instance $\bowtie(\neg s, s)$ does not yield any substitution and, therefore, the match in such a case does not hold.

[1] The QoS value of a path $s_0 \overset{\kappa_1}{\frown} s_1 \ldots s_{n-1} \overset{\kappa_n}{\frown} s_n$ is defined as $\kappa_1 \cdot \ldots \cdot \kappa_n$.

4 Examples

In this section we present some specification examples. To make the presentation more readable let us introduce some notational conventions. First, we avoid writing trailing **0** processes, second, we write $\varepsilon[P]@r$ instead of $\varepsilon_{\mathbf{1}}[P]@r$ and similarly for $node_{\mathbf{1}}\langle t\rangle$, $con_{\mathbf{1}}\langle t\rangle$ and $acc_{\mathbf{1}}\langle t\rangle$.

Boolean expressions Booleans are encoded as processes that allocate a pair of names to a node:

$$\begin{aligned} True\ r &\stackrel{\text{def}}{=} (\nu\, t)\varepsilon[\langle t, t\rangle]@r \\ False\ r &\stackrel{\text{def}}{=} (\nu\, f, f')\varepsilon[\langle f, f'\rangle]@r. \end{aligned}$$

The truth and the falsity are tested by checking that the names in a pair are equal or different, respectively. The following process tests for the equality of two names:

$$Test\ x\ y\ r \stackrel{\text{def}}{=} (\nu\, t)(node\langle t\rangle.\varepsilon[Eval\ y\ r \ \mid\ \langle x\rangle]@t),$$

where $Eval\ y\ r \stackrel{\text{def}}{=} (y).True\ r \ \mid\ (\neg y).False\ r$. Process *Test* spawns the tuple $\langle x\rangle$ and the *Eval* process onto a newly generated node so that the first or the second component of *Eval* have exclusive access to $\langle x\rangle$. Notice that only one of the components can consume the tuple, indeed, either $x = y$ (and only the pattern (y) matches $\langle x\rangle$) or $x \neq y$ (and only the pattern $(\neg y)$ matches $\langle x\rangle$). Finally, *True* or *False* allocates on node r the truth value corresponding to evaluation of $x = y$. Assuming the encoding of booleans, we can represent standard control structures such as `if-then-else` and `while`.

The encoding of boolean values is indeed an example of a standard programming metaphor for finding and handling services. Assume that we want to describe a *look-up* mechanism for discovering distributed services. For instance, the *web services* technology allows deploying new services by gluing together those that have been published. Web service composition, however, requires a look-up phase where the available service must be discovered. In the boolean example, processes *True* and *False* are the services that have been published and composed together to provide the *Test* service. Notice that the look-up phase does not require the knowledge of the service name but only that of the "schema" of the service. For instance, whenever a new "true" service is published it suffice to generate a new name and use it for building the "schema" for the true service (i.e., a pair of two equal names).

Public, private, permanent and stable links Links in $\mathcal{K}oS$ are public entities: when available they can be exploited by all processes. Consider the following $\mathcal{K}oS$ net:

$$N \stackrel{\text{def}}{=} s :: \varepsilon_3[P]@t \ \parallel\ s \overset{1}{\frown} r \ \parallel\ r :: con_2\langle t\rangle.\varepsilon_2[Q]@t \ \parallel\ t :: acc_2\langle r\rangle,$$

where QoS values are the c-semiring of natural numbers. Net N has three nodes s, r and t and, initially, only s and r are connected by a link with QoS value 1. Node s is trying to spawn P on t which is not possible because there is no path from s to t. Node r is willing to spawn a process Q on t, as well; however, r is aware that a link must be

first created. Node t simply accepts requests for establishing a link from r. Initially, it is only possible to synchronize $con_2\langle t\rangle$ and $acc_2\langle r\rangle$ which, by applying rule ( ) leads to

$$N' \stackrel{\text{def}}{=} s :: \varepsilon_3[P]@t \quad \| \quad s \overset{1}{\frown} r \quad \| \quad r :: \varepsilon_2[Q]@t \quad \| \quad r \overset{2}{\frown} t \quad \| \quad t :: \mathbf{0}.$$

Now, applying rules ( ), ( ) and ( ) we derive

$$N' \xrightarrow[2]{\tau} s :: \varepsilon_3[P]@t \quad \| \quad s \overset{1}{\frown} r \quad \| \quad r :: \mathbf{0} \quad \| \quad t :: Q.$$

Notice that the link between r and t is consumed by the migration of Q hence P cannot reach t. However, N' can also evolve differently, in fact, both the two spawning actions are enabled, because the creation of the link between r and t has also provided a path from s to t exposing the QoS value 3. Hence, by rules ( ), ( ), ( ) and ( ) we can also derive

$$N' \xrightarrow[3]{\tau} s :: \mathbf{0} \quad \| \quad r :: \varepsilon_2[Q]@t \quad \| \quad t :: P.$$

Noteworthy, the migration of P prevents Q to be spawned because the link created by r has been used by P.

In general, this kind of interference should be avoided and this can be done in $\mathcal{K}oS$ by expressing *private links* which can be specified by exploiting the properties of c-semirings. The intuition is that the use of a link is allowed only whether the traversing process has the appropriate "rights". If we represent access rights as sets of names, then a process must "know" all the names needed for traversing the link. For instance, consider the following net:

$$s :: \varepsilon_{\{r,s\}}[P]@t \parallel s \overset{\{r\}}{\frown} s',$$

process P can traverse the link $s \overset{\{r\}}{\frown} s'$ because it "knows" r, that is the only name required to traverse the link. Noteworthy, P could not traverse $s \overset{\{r,u\}}{\frown} s'$ because it does not expose name u.

We consider the c-semiring $\mathcal{R} = \langle \wp_{\text{fin}}(\mathcal{S}) \cup \{\mathcal{S}\}, glb, \cup, \mathcal{S}, \emptyset\rangle$ to represent access rights (recall that $\mathcal{S}$ is the set of sites). It is straightforward to prove that $\mathcal{R}$ is a c-semiring; moreover, the order induced by the additive operation of $\mathcal{R}$ is the inverse of the set inclusion (i.e., $X \leq Y \iff Y \subseteq X$).

Therefore, a private link between the nodes s and t can be specified as

$$(\nu\, p)(s :: P \parallel s \overset{\{p\}}{\frown} t \parallel t :: Q),$$

indeed, in order to pass through link $s \overset{\{p\}}{\frown} t$, a process must exhibit the "password" p. The knowledge of p is handled by enlarging the scope of the restriction and communicating it.

We conclude by illustrating how one could implement *permanent* links, i.e. links that are always available, by exploiting replication:

$$s :: !con_\kappa\langle t\rangle \parallel t :: !acc_{\kappa'}\langle s\rangle$$

A slight variation are *stable* links, which are links existing until a given condition is satisfied.

$$Stable_s\ G\ t \stackrel{\text{def}}{=} !con_\kappa\langle t\rangle \quad | \quad \varepsilon[While\ G\ do\ acc_\kappa\langle s\rangle\ od\ \mathbf{0}]@t$$

Cryptography By exploiting private links, $\mathcal{K}oS$ can encode standard encryption/decryption mechanisms usually adopted for expressing security protocols in process calculi (see e.g. [1]). Consider the following net:

$$(\nu\, k, s_k)(i :: P \parallel i \overset{\{k\}}{\frown} s_k \parallel s_k :: M \parallel s_k \overset{\{k\}}{\frown} r \parallel r :: Q), \tag{1}$$

and assume that the only links from/to s_k are those appearing in (1). Net (1) aims at representing the initiator i and the responder r of a protocol that share a key k. According to (1) a key is modelled by means of a pair made of a name and a node which roughly speaking contains those messages that are encrypted with k.

The intuition is that encrypting corresponds to allocating a message on s_k while decrypting corresponds to the possibility of "jumping" on s_k and reading a message or, in other words, to the knowledge of k for traversing links $i \overset{\{k\}}{\frown} s_k$ or $s_k \overset{\{k\}}{\frown} r$.

5 Composing Overlay Networks

We consider a scenario where a service is replicated over the nodes of an overlay network and can be invoked trough a *unique* handler H that manages the requests of the clients. This kind of architectures is adopted from Internet Service Providers (ISP) that offer dial-up connection to end-users (EU). In this case a telecommunication company (TC) handles the phone overlay networks. The EU connects to the "nearest" ISP server by dialing a single (country-wide) number. The TC takes care of dispatching the call to the closest ISP server on the overlay network. There are (at least) two possible way of connecting the EU and the ISP server. Either the TC establishes a direct connection between the EU and the ISP, or the TC act as a gateway between the phone overline network and the ISP overlay network. Both solutions can be easily expressed in $\mathcal{K}oS$ in the logical architecture of the system: the handler H manages the requests (e.g., controls the access rights of the client), looks for a suitable server, and forwards the request, while trying to balance the load of any replica of the server. Hence, the request of a client C might not be forwarded to the "best" server from the client's point of view. In this case, H provides another server to C, however, the client may or may not commit to use it.

The simplest way to model this composed overlay network is to assume that the link between C and H have QoS values expressing the access rights of C. When a server s meeting both the request of C and the load constraints is found, H replies to C and tells s to accept a (private) link from C. Hereafter, we assume that h is the node address of H. We detail the client first:

$$\begin{aligned} C\ \kappa\ pr\ c \overset{\text{def}}{=}\ (\nu\, r)(\ & \varepsilon_\kappa[\langle \text{“connect”}, c, r\rangle]@h. \\ & (r, ?s, ?pr', ?p). \\ & \mathit{If}_c\ pr' < pr \\ & \quad \mathit{then}\ con_{\{r,p\}}\langle s\rangle.\varepsilon_{\{p,r\}}[R]@s \\ & \quad \mathit{else}\ con_{\{r,p\}}\langle s\rangle.\varepsilon_{\{p,r\}}[\langle r, \text{“to-much”}\rangle]@s). \end{aligned}$$

Process C requests H to find a server and waits for the response. The request contains c, the node address of C, and a private name r. Name r can be thought of as the unique

marker of the request so that only C will acquire data corresponding to request r. Possibly, H returns a response (marked with r) containing the server address s, the price pr', and the password p for the private link. Finally, C establishes a private link with s and, depending on the price pr' required by the server, either raises its request R ('then' branch) or notifies s that the service is too expensive ('else' branch).

The definition of H requires the following auxiliary processes.

$$Rd(T) \stackrel{\text{def}}{=} (T).\langle \mathfrak{t}_T \rangle$$
$$Lt_s\ r\ i \stackrel{\text{def}}{=} Rd(r, ?j).If_s\ j \leq i\ then\ True\ else\ False$$

Process $Rd(T)$ looks for a tuple matching T and immediately re-generates the consumed tuple; this is denoted by $\mathfrak{t}_T$ which is obtained from T by removing all the '?' occurring in its placeholders. Then, process $Lt_s\ r\ i$, interpreting $\langle r, v \rangle$ as a "cell" having address r and containing value v, reads the value in r and establishes if it is less than/equal to i.

$$\begin{array}{l}
H \stackrel{\text{def}}{=} !(\ (\text{“connect”}, ?x, ?r).\langle r, 1\rangle \\
\quad While_h\ Lt_h\ r\ \text{nserv} \\
\quad\quad do \\
\quad\quad\quad (r, ?i).Rd\ (pref(x, i), ?l, ?pr.) \\
\quad\quad\quad If_h\ l \geq \max \\
\quad\quad\quad\quad then\ \langle r, i+1\rangle \\
\quad\quad\quad\quad else \\
\quad\quad\quad\quad\quad (\nu\ p)(\varepsilon[\langle \text{“newlink”}, x, r, p\rangle]@pref(x, i). \\
\quad\quad\quad\quad\quad\quad\quad \varepsilon[\langle r, pref(x, i), pr, p\rangle]@x. \\
\quad\quad\quad\quad\quad\quad\quad (x, ?v).acc_{f(v,l)}\langle x\rangle.\langle x, f(v, l)\rangle. \\
\quad\quad\quad\quad\quad\quad\quad (pref(x, i), ?l, ?pr).\langle pref(x, i), l+1, pr\rangle) \\
\quad\quad od\ \varepsilon[\langle r, \text{“no-server-available”}\rangle]@x).
\end{array}$$

Process H is continuously listening for a connection request. Once such a request is issued from a client at x, H starts scanning the server list (nserv is the number of servers). For each server s, node h contains a tuple $\langle s, l, pr\rangle$ where l is an estimation of the load of s and pr is the price for using s. Also, for each client x, h maintains a tuple $\langle x, \kappa\rangle$ that reports the connection between H and x (as done in Section 2). Moreover, H uses a function $pref$ that, given the client address x and the index i, yields the i-th server "preferred" by the client. At the i-th iteration of the while loop, H reads the information of the i-th server preferred by x and, if the load of such a server is too high, the cycle is repeated provided that more servers are left ('then' branch); otherwise, a password p for a private link is generated and communicated to both x and the selected server. The server will accept a private link creation from x so that the client owning the password p can perform a request at s. Finally, H re-establish a link with x according to the new load of the servers by exploiting function f and reflecting this changes in the tuple corresponding to x (i.e., $\langle x, f(v, l)\rangle$), as in Section 2. Indeed, the mechanism of load balancing is the one defined in Section 2, the only difference being that now H is the unique handler that manages the connections with the clients.

Given H and C, the servers must simply wait for a connection request (issued from H) and establish the private connection with the client:

$$S\ s \stackrel{\text{def}}{=} !(\ (\text{"newlink"}, ?x, ?r, ?p).acc_{\{r,p\}}\langle x\rangle.\text{....wait \& execute....}\\ \varepsilon[(s, ?l, ?pr).\langle s, l-1, pr\rangle)]@h)).$$

Once the request has been served, S simply updates the load of s.

A net where C, H and S work can be defined as follows.

$$\begin{array}{c} \|_{i=1,\ldots,m}\ x_i :: C\ \kappa_i\ pr_i\ x_i \quad | \quad !acc\langle h\rangle \quad | \quad con_{\kappa_i'}\langle h\rangle \\ \|_{i=1,\ldots,m}\ h :: !con\langle x_i\rangle \ \|\ h :: H\ \| \\ \|\ (\nu\ s_1, \ldots, s_n)(\ \|_{j=1,\ldots,n}\ h :: !con\langle s_j\rangle \quad | \quad \langle s_j, l_j, pr_j\rangle \ \|\ s_j :: S\,) \end{array}$$

where $\|_{i=1,\ldots,m} N_i$ shortens $N_1 \ \|\ \ldots\ \|\ N_n$.

The other solution touched upon at the beginning of this section can be achieved by exploiting the possibility offered by $\mathcal{K}oS$ of "connecting" links to form paths between nodes. More precisely, instead of connecting directly the client's node x and (the node of) the selected server s, we can connect h and s so that the client's request at s is routed through h.

6 Conclusion

We have formally defined $\mathcal{K}oS$ a process calculus that provides basic primitives to describe QoS requirements of distributed applications. We demonstrated the applicability of the approach by specifying some expressive case studies.

Our research program is to provide a solid foundation to drive the design of languages and middleware having application-oriented QoS mechanisms. The work reported here is a preliminary step in this direction. In terms of calculus design, the current definition of $\mathcal{K}oS$ assumes that links are the basic construct to manage QoS interactions and cooperation. This is a reasonable assumption for several cases. For instance, in this paper we handled the QoS composition between different overlay networks by suitable links. However, one could interpret QoS composition of overlay networks in a more general sense than adding suitable links. An interesting challenge for future research is to extend $\mathcal{K}oS$ with more general mechanisms for composing overlay networks than simple parallel composition via links.

There are a number of ways in which our setting can be extended. For instance, it would be interesting to develop type systems which would allow determining QoS properties of processes. We plan to extend types for access control of [7, 6] to deal with QoS attributes. In particular, it would be interesting to exploit such types to capture the notion of contract. Another direction for future research is developing observational semantics for $\mathcal{K}oS$ based on the idea of observing QoS values. These abstract theories could permit reasoning on $\mathcal{K}oS$ nets and comparing them on the basis of the perceived QoS values.

References

1. M. Abadi and A. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1):1–70, January 1999.
2. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, March 1997.
3. L. Cardelli and D. Rowan. Service combinators for web computing. *Software Engineering*, 25(3):309–316, 1999.
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 *Lecture Notes in Computer Science*, pages 436 – 479. Springer-Verlag, 2003.
5. R. De Nicola, G. Ferrari, and R. Pugliese. KLAIM: A kernel language for agents interaction and mobility. *IEEE/ACM Transactions on Networking*, 24(5):315–330, 1998.
6. R. De Nicola, G. Ferrari, and R. Pugliese. Programming access control: The KLAIM experience. In *International Conference in Concurrency Theory*, Lecture Notes in Computer Science. Springer-Verlag, 2000.
7. R. De Nicola, G. Ferrari, R. Pugliese, and B. Venneri. Types for access control. *Theoretical Computer Science*, 240(1):215–254, June 2000.
8. G. Ferrari, U. Montanari, and E. Tuosto. Graph-based Models of Internetworking Systems. In T. Aichernig, Bernhard K. Maibaum, editor, *Formal Methods at the Crossroads: from Panaces to Foundational Support*, volume 2757 of *Lecture Notes in Computer Science*, pages 242 – 266. Springer-Verlag, 2003.
9. A. Igarashi and N. Kobayashi. A generic type system for the Pi-calculus. *Theoretical Computer Science*, 311(1–3):121–163, Jan. 2004.
10. N. Kobayashi. Type Systems for Concurrent Processes: From Deadlock-Freedom to Livelock-Freedom, Time-Boundedness. In J. van Leeuwen, O. Watanabe, M. Hagiya, P. D. Mosses, and T. Ito, editors, *Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics, Proceedings of the International IFIP Conference TCS 2000 (Sendai, Japan)*, volume 1872 of *Lecture Notes in Computer Science*, pages 365–389. IFIP, Springer-Verlag, Aug. 2000.
11. G. Meredith and S. Bjorg. Service-Oriented Computing: Contracts and Types. *Communications of the ACM*, 46(10):41 – 47, October 2003.
12. M. Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata Languages and Combinatorics*, 7(3):321–350, 2002.
13. U. Nestmann and R. Fuzzati. Unreliable failure detectors with operational semantics. In *Proc.ASIAN 2003*, Lecture Notes in Computer Science. Springer-Verlag, 2003.
14. M. Papazouglou and D. Georgakopoulos. Special issue on service oriented computing. *Communications of the ACM*, 46(10), 2003.
15. C. Priami. Stochatic π-calculus. *The Computer Journal*, 38(6):578–589, 1995.
16. G. Theodorakopoulos and J. Baras. Trust Evaluation in AdHoc Networks. In *WiSe '04: Proceedings of the 2004 ACM workshop on Wireless security*, pages 1–10. ACM Press, 2004.
17. E. Tuosto. *Non-Functional Aspects of Wide Area Network Programming*. PhD thesis, Dipartimento di Informatica, Università di Pisa, May 2003. TD-8/03.