Reputation-based Composition of Social Web Services

Alessandro Celestini[1], Gianpiero Costantino[2], Rocco De Nicola[1], Zakaria Maamar[3], Fabio Martinelli[2], Marinella Petrocchi[2], and Francesco Tiezzi[1]

[1] IMT Institute for Advanced Studies, Lucca, Italy
{alessandro.celestini,rocco.denicola,francesco.tiezzi}@imtlucca.it

[2] CNR, Istituto di Informatica e Telematica, Pisa, Italy
{gianpiero.costantino,fabio.martinelli,marinella.petrocchi}@iit.cnr.it

[3] Zayed University, Dubai, U.A.E.
zakaria.maamar@zu.ac.ae

— November 2, 2013 —

Abstract. Social Web Services (SWSs) constitute a novel paradigm of service-oriented computing, where Web services, just like humans, sign up in social networks that guarantee, e.g., better service discovery for users and faster replacement in case of service failures. In past work, composition of SWSs was mainly supported by specialised social networks of competitor services and cooperating ones. In this work, we continue this line of research, by proposing a novel SWSs composition procedure driven by the SWSs reputation. Making use of a well-known formal language and associated tools, we specify the composition steps and we prove that such reputation-driven approach assures better results in terms of the overall quality of service of the compositions, with respect to randomly selecting SWSs.

Keywords: Social Web Services, Reputation Systems, Stochastic Analysis

1 Introduction

In line with previous research, e.g., [8, 17, 20] and [21], this paper studies the blend of social computing (exemplified by social networks) with service-oriented computing (exemplified by Web services). When the latter joins the former, we obtain *Social Web Services* (SWSs). Social Networks (SNs) illustrate the willingness of users to share information, work with others, and recommend services and applications. These various and rich forms of interactions are built upon the basic principles of "I offer services that somebody else may need" and "I require services that somebody else may offer", principles that regulate Web services operation too. Following this intuition and comparing SWSs to regular Web services, SWSs establish and maintain networks of contacts, count on their contacts when needed, form strong and long lasting collaborative social groups.

To support a SWS to carry out these operations, past literature relies on social networks of collaboration, substitution, and competition in which SWSs sign up and thus, become members. A SWS uses these networks to identify the services that it likes to work with in case of composition, to identify the services that can replace it in case of failure when it participates in a composition, and to be aware of the services that compete against it in case of selection.

In this work, we focus on competition and collaboration SNs: as proposed in [18], collaborations and competitions are social networks of SWSs that are built to support the development of composite Web services. In particular:

- **Competition Social Networks**. Web services that expose similar functionalities belong to these networks. They are in competition against each other when a SWS has to be selected to perform a specific task within a service composition. Typically, competitor SWSs are chosen according to some quality valued they expose, quantifying both business and security guarantees (see, e.g., [22]).
- **Collaboration Social Networks**. This kind of networks involve Web services that expose different functionalities. By being part of a collaboration SN, a SWS knows the services that it likes to work with when the time for developing compositions comes.

Building upon competing and collaborating SWSs, we aim at formally modelling and analysing a composition of SWSs, where SWSs selection is based on their *reputation*. Reputation systems are used as decision support systems to drive interactions between parties in a networked system. The basic assumption is that for each party is possible to store the information related to party's past interactions. On the basis of such information a reputation score is computed for each party. Thus, when a party has to choose another party to interact with, it makes the decision on the basis of parties' reputation score. Usually, higher the reputation score, higher the probability that a party will be selected for a future interaction. In fact, a high reputation score means that the party is trustworthy (in terms of provided QoS) and thus the risk when interacting with it will be low. In this work we use reputation scores to drive SWSs composition.

Contribution With this paper, we envisage a twofold contribution. On the one hand, we make use of formal specification languages and analysis tools to model a novel procedure for SWSs composition and prove properties related to the overall quality of the composition result. To the best of our knowledge, this is the first attempt to analyse QoS properties of the composed social Web services making use of such formal languages and tools. On the other hand, we continue the research effort to aid societies and enterprises to benefit and capitalise on Web 2.0 applications [5], matching together social networks, web services, and reputation systems, the latter as the novel factor to automatically drive formation of high-quality social Web service compositions.

Roadmap This paper is structured as follows. Section 2 discusses related work. In Section 3, we describe how to compose SWSs by relying on their reputation. Section 4 recalls the formal methods we use for specifying and analyzing the scenario associated with SWSs composition. Section 5 provides the formal specification of the SWSs composition. In Section 6, we analyse properties of the specification. Finally, Section 7 concludes the paper.

2 Related Work

We recall research approaches on the combination of social computing and service-oriented computing.

Work in [8, 17, 20] and [21], deploys SNs of SWSs to address certain issues like Web services discovery. In [8], Chen and Paik build a global social service-network to improve service discovery. The authors link services together using specific data correlations. In [17], Maamar et al. develop a method to engineer SWSs. Questions that the method addresses include: what relations between Web services exist, what SNs correspond to these relations, how to build SNs of SWSs, and what social behaviors SWSs can exhibit. Last but not least, Maamar et al. use SNs of SWSs to tackle the "thorny" problem of Web services discovery [20]. At run-time Web services run into various situations like competing against similar peers during selection, collaborating with different peers during composition, and replacing similar Web services during failure despite the competition. These situations help build the privileged contacts of a SWS.

Work in [1, 4, 24–26] and [27] deploys SNs of persons using Web services as an implementation technology. In particular, in [1], Al-Sharawneh and Williams mix semantic Web, social networks, and recommender systems to assist users selecting Web services with respect to their functional and non-functional requirements. In [4], Bansal et al. examine trust for Web services discovery. Users' trust in Web services' providers is the social element in this discovery. In [24], Maaradji et al. propose a social composer (aka *SoCo*) that provides advices to users on what actions they need to take in response to specific events like selecting specific Web services. Wu et al. rank Web services based on their popularity among users [26].

A community exists that mixes SNs of users and SNs of SWSs, Maamar et al. intertwine these two categories of networks to compose, execute, and monitor composite Web services [18]. To achieve this intertwine, three components are developed: composer, executor, and monitor. The social composer develops composite Web services considering relations between users and between Web services. The social executor assesses the impact of these relations on the execution progress of these composite Web services. Finally, the social monitor replaces failing Web services to guarantee the execution continuity of these composite Web services.

With respect to the works on SWSs discussed so far, our work differs for a synergistic usage of reputation systems and social networks. Reputation of Web services is based on the Quality of Service (QoS) perceived by their users and is exploited to support selection decisions in SWSs compositions. This reputation-based selection strategy, on the one hand, permits removing non-deterministic or user-driven choices, which results in a more automatable approach. On the other hand, as demonstrated by the formal analysis illustrated in Sections 5 and 6, this approach ensures, in general, better performance in terms of compositions' QoS.

Concerning techniques and tools for analysing distributed systems, among the many works proposed in the literature we mention those regarding the stochastic instruments we have exploited for our analysis: StoKlaim, MoSL and SAM. [9] introduces the stochastic specification language StoKlaim and the stochastic logic MoSL. [6] presents the analysis tool SAM, which permits checking properties specified in MoSL against StoKlaim specifications. In particular, the tool is used there to model and analyse three

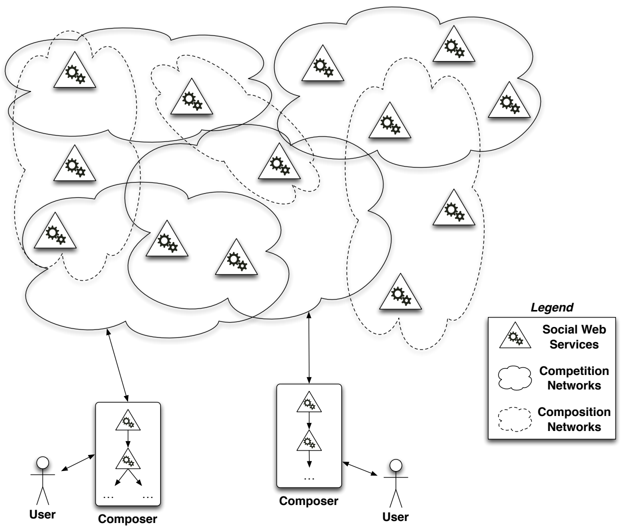

Fig. 1: Social Web Service scenario

classical leader election algorithms. Finally, [14] and [7] present the application of this stochastic verification methodology and related tools to two different domains: collective robotic systems and reputation systems, respectively. In both cases, simulation and model checking results are discussed.

3 Reputation-based Composition of Social Web Services

This section describes a reputation-based procedure to compose SWSs. During composition, selection is also aided by competition and collaboration SNs. Such social composition is partly inspired by the work in [23]. As a novel factor, we introduce here the SWSs reputation as the metric to select the high quality Web services within a set of collaboration and competition networks. This should lead to higher quality compositions, as demonstrated by the analysis described in Sections 5 and 6.

In the scenario shown in Fig. 1, Users put together a composite service by selecting the appropriate SWSs via the two kinds of SNs. In order to be part of a composition, SWSs receive composition requests from Composers. A Composer is a software entity, activated by a User, that is in charge of discovering and querying a pool of SWSs suitable for the desired composition.

When a SWS receives a composition request, it can either accept or reject such request [19]. The SWS makes such decision based on its current load (e.g., availability, level of commitment in other compositions and so on). Notably, the service can accept and serve more requests simultaneously, up to a given threshold. If it accepts the composition request, it activates a new *transaction* (i.e., a service instance) waiting for the service request coming from the user's composer.

A transaction is completed, with a given QoS, once the service is provided to the user. According to such QoS, the transaction of a SWS is rated by the user. Such rat-

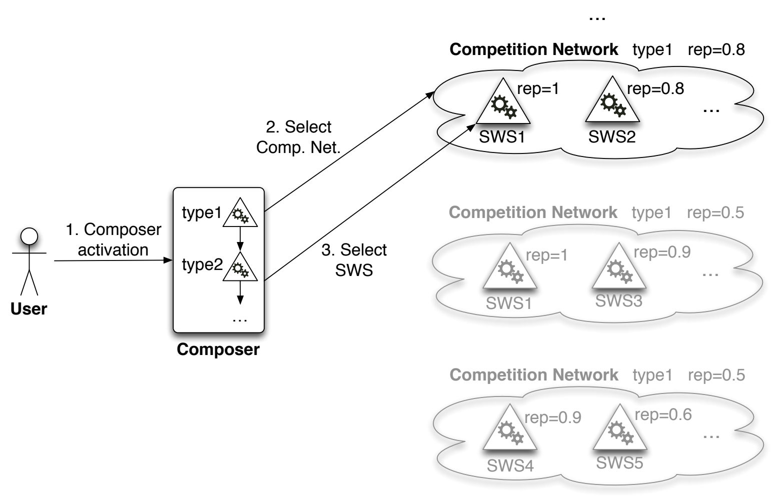

Fig. 2: Reputation-based composition: initial steps

ings are then aggregated to compute the reputation score of SWSs. Different reputation models can be used to aggregate ratings (in this work, we rely on the Beta model [15]). The reputation of an SWS is therefore an estimation of its QoS based on its past interactions.

The SWSs' reputation scores can be aggregated in order to obtain the overall reputation of social networks of SWSs. Also in this case, depending on the considered application domain, different forms of aggregation can be used: average of reputations (possibly excluding highest and lowest values), minimum reputation, maximum reputation, etc.

In the following, we describe the various steps of the reputation-based composition. We assume that each service offered in this scenario is free of charge.

First, each User chooses the desired SWSs composition and activates the corresponding Composer (Figure 2, step 1). Then, the Composer manages all transactions required by the service composition. When the composed service completes, the User evaluates the overall QoS of the composition according to the QoS of all sub-services.

Secondly, the Composer works by selecting a competition network whose members are SWSs providing the same service type of the first service in the composition (Figure 2, step 2). The network selection is driven by the overall reputation of all the competition networks available, whose members provide that service type.

The Composer selects the competition network with the highest reputation score. Among the network members, the Composer sends a composition request to the SWS with the highest reputation (Figure 2, step 3). If this service rejects the request, the Composer selects the SWS with the highest reputation (of course, not considering the previous service) belonging to one of the competition networks whose the previous SWS is a member (Figure 3). It proceeds in this way until it finds an available service.

Notably, we assume that a SWS is member of a restricted number of competition networks; indeed, a high quality SWS will tend to only belong to networks with high reputation in order to avoid compositions with low quality services. It is worth notic-

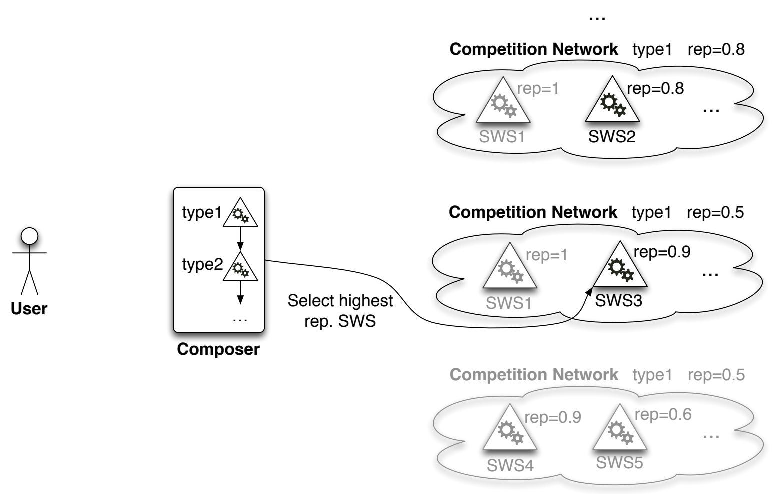

Fig. 3: Reputation-based composition: in case SWS1 rejects, selection of the SWS with highest reputation in the competition networks of SWS1

ing that selection from a competition network should guarantee a faster discovery than considering larger sets of SWSs.

When a SWS accepts the composition request, the Composer sends the service request, receives the service, evaluates its quality (i.e., rates it) on behalf of the User, and starts searching the next SWS. After the first service completion, the subsequent services are identified by resorting to the collaboration networks of the previous SWS (Figure 4). Analogously to the previous case, the selection of the SWS is based on the type of the service and on its reputation. If the chosen SWS rejects the composition request, its competition networks are exploited to find an alternative SWS of the same type, as discussed above. When the Composer receives the service from the last SWS in the composition, it notifies the User, who will evaluate the overall composition.

It is worth noticing that, as in [23], Composers follow a sequential approach in order to use the SNs of the last run service in the composition.

4 Klaim and Related Stochastic Verification Tools

In this section, we provide a brief overview of the formal methods we use for specifying and analysing the scenario associated with SWSs compositions: the coordination language Klaim [10] (and its stochastic extension [9]), the stochastic logic MoSL [11] and the analysis tool SAM [16].

4.1 Specification

Klaim is a tuple-based coordination language specifically designed for modelling mobile and distributed applications and their interactions, which run in a network environment.

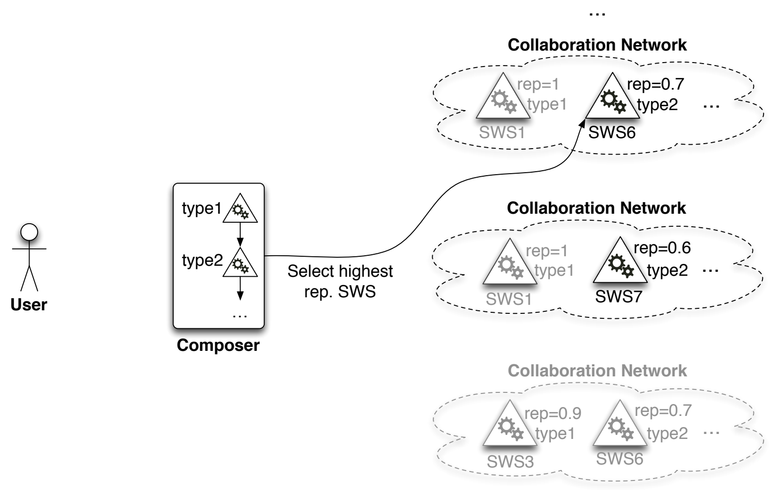

Fig. 4: Reputation-based composition: in case SWS1 accepts, selection of the SWS of type2 with highest reputation in the collaboration networks of SWS1

In our presentation of Klaim we consider a version of the language enriched with standard control flow constructs (i.e., assignment, if-then-else, sequence, etc.). Such constructs were not included in the original presentation of the language [10], however they can be easily rendered in Klaim (by resorting, e.g., to choice, fresh names and recursion in the usual way) and are directly supported by related tools (and, in particular, by the analysis tool used in this work).

The syntax we use is reported in Table 1, where s, s',... range over *locality names* (i.e., network addresses); **self**, l, l',... range over *locality variables* (i.e., aliases for addresses); ℓ, ℓ',... range over locality names and variables; x, y,... range over *value variables*; X, Y,... range over *process variables*; e, e',... range over *expressions*[4]; A, B,... range over *process identifiers*[5]. We assume that the set of variables (i.e., locality, value and process variables), the set of values (locality names and basic values), and the set of process identifiers are countable and pairwise disjoint.

Klaim specifications consist of *nets*, namely finite plain collections of nodes where *components*, i.e. processes and data tuples, can be allocated. Nodes are composed by means of the parallel composition operator $\_\|\_$. At net level, it is possible to restrict the visibility scope of a name s by using the operator $(\nu s)\_$: in a net of the form $N_1 \| (\nu s)N_2$, the effect of the operator is to make s invisible from within the subnet N_1.

Nodes have the form $s\ ::_{\rho}\ C$, where s is a unique locality name ρ is an allocation environment, and C is a set of hosted components. An *allocation environment* provides a

[4] The precise syntax of expressions is not specified here. Suffice it to say that expressions contain basic values (booleans, integers, strings, floats, etc.) and variables, and are formed by using the standard operators on basic values.

[5] We assume that each process identifier A with arity n has a unique definition, visible from any locality of a net, of the form $A(f_1, \ldots, f_n) \triangleq P$, where formal parameters f_i are pairwise distinct.

(*Nets*)	$N ::= \mathbf{0} \;\big\vert\; s ::_{\rho} C \;\big\vert\; N_1 \parallel N_2 \;\big\vert\; (\nu s)N$
(*Components*)	$C ::= \langle t \rangle \;\big\vert\; P \;\big\vert\; C_1 \mid C_2$
(*Processes*)	$P ::= \mathbf{nil} \;\big\vert\; a \;\big\vert\; P_1 \,; P_2 \;\big\vert\; P_1 \mid P_2 \;\big\vert\; \mathbf{if}\ (e)\ \mathbf{then}\ \{P\}\ \mathbf{else}\ \{Q\}$ $\;\big\vert\; \mathbf{for}\ i = n\ \mathbf{to}\ m\ \{\ P\ \} \;\big\vert\; \mathbf{while}\ (e)\ \{P\} \;\big\vert\; A(p_1, \ldots, p_n)$
(*Actions*)	$a ::= \mathbf{in}(T)@\ell \;\big\vert\; \mathbf{read}(T)@\ell \;\big\vert\; \mathbf{out}(t)@\ell \;\big\vert\; \mathbf{eval}(P)@\ell \;\big\vert\; x := e$ $\;\big\vert\; \mathbf{newloc}(s)$
(*Tuples*)	$t ::= e \;\big\vert\; \ell \;\big\vert\; P \;\big\vert\; t_1, t_2$
(*Templates*)	$T ::= e \;\big\vert\; \ell \;\big\vert\; P \;\big\vert\; !\,x \;\big\vert\; !\,l \;\big\vert\; !\,X \;\big\vert\; T_1, T_2$

Table 1: Klaim syntax

name resolution mechanism by mapping locality variables l, occurring in the processes hosted in the corresponding node, into localities. The distinguished locality variable **self** is used by processes to refer to the address of their current hosting node. In the rest of this section, we will use ℓ to range over locality names and variables.

Processes are the Klaim active computational units and may be executed concurrently either at the same locality or at different localities. They are built up from the process **nil**, which does nothing, from the process calls $A(p_1, \ldots, p_n)$, where p_i denote actual parameters, and from basic actions by means of sequential composition $\_;\_$, parallel composition $\_ \mid \_$, conditional choice **if** (e) **then** $\{\_\}$ **else** $\{\_\}$, the iterative constructs **for** $i = n$ **to** m $\{\_\}$ and **while** (e) $\{\_\}$.

During their execution, processes perform some basic *actions*. Actions $\mathbf{in}(T)@\ell$ and $\mathbf{read}(T)@\ell$ are retrieval actions and permit to withdraw/read *data tuples* (i.e. sequences of values) from the tuple space hosted at the (possibly remote) locality ℓ: if a matching tuple is found, one is non-deterministically chosen, otherwise the process is blocked. These actions exploit templates as patterns to select tuples in shared tuple spaces. *Templates* are sequences of actual and formal fields, where the latter are written $!\,x$, $!\,l$ or $!\,X$ and are used to bind variables to values, locality names or processes, respectively.

Action $\mathbf{out}(t)@\ell$ adds the tuple resulting from the evaluation of tuple t (which may contain expressions) to the tuple space of the target node identified by ℓ, while action $\mathbf{eval}(P)@\ell$ sends the process P for execution to the (possibly remote) node identified by ℓ. Actions **out** and **eval** are both non-blocking.

Finally, action **newloc** creates new network nodes, while action $x := e$ assigns the value of e to x. These latter two actions, differently from all the others, are not indexed with an address because they always act locally.

4.2 Klaim's Related Tools

We introduce now the stochastic analysis tools of Klaim, that enable us to perform quantitative analysis of systems. In general, two main kinds of analysis can be performed over systems, quantitative or qualitative. In qualitative analysis it is verified whether a certain event will occur. In quantitative analysis, instead, it is verified what is the probability that a certain event will occur. In order to perform quantitative analysis of a KLAIM specification, we have to enrich the formalism by enabling the modelling of random phenomena. KLAIM specifications can be enriched with stochastic aspects, using the KLAIM's stochastic extension STOKLAIM, while the properties of the considered system can be expressed by using the stochastic logic MoSL. The properties of interest are then checked against the STOKLAIM specifications by means of the analysis software tool SAM. We provide below a very short overview of STOKLAIM, MoSL and SAM.

STOKLAIM In STOKLAIM [11], KLAIM's process actions are enriched with a rate. Such rate is the parameter of an exponentially distributed random variable characterising the duration of the execution of an action. In particular, such random variables are governed by a negative exponential distribution. This distribution is related to the Poisson distribution and describes the time between events in a Poisson process, i.e. a process in which events occur continuously at a constant average rate λ, independently of the time t. A real valued random variable X has a *negative exponential distribution* with rate $\lambda > 0$ if and only if the probability that $X \leq t$ with $t > 0$ (i.e., the probability that an event occurs within t time units) is $1 - e^{-\lambda \cdot t}$. The expected value of X is λ^{-1}, while its variance is λ^{-2}. The use of the exponential distribution is motivated by the fact that it enjoys convenient properties enabling automated analyses that are not always allowed by other distributions.

The operational semantics of STOKLAIM permits associating to each specification a Continuous Time Markov Chain (CTMC), one of the most popular models for the evaluation of the performance and dependability of information processing systems. Such CTMC is then used to perform quantitative analyses of the considered system.

MoSL The desired properties of a system under verification are formalised using the mobile stochastic logic MoSL [11]. MoSL formulae use predicates on located tuples and express reachability of states of interest, while passing through, or avoiding, other intermediate states.

MoSL builds on CTL [13] and CSL [2, 3] but incorporates the basic features of MoMo [12] to exploit the distributed features of systems. Indeed two key operators of MoSL are variants of the MoMo consumption ($\rightarrow$) and production ($\leftarrow$) operators.

Intuitively, a Klaim net satisfies a consumption formula $A(p_1, ..., p_n)@\ell \rightarrow \Phi$ if it contains a process A running at site ℓ, and the remaining network, namely the context in which $A(p_1, ..., p_n)$ is operating, satisfies Φ. Similarly, formula $\langle T \rangle @\ell \rightarrow \Phi$ holds whenever a tuple t matching T is stored at site ℓ, and the remaining network satisfies Φ. A production formula $\langle t \rangle @\ell \leftarrow \Phi$ holds if the network satisfies Φ whenever tuple t is stored in a node of site ℓ. In particular, the satisfaction of Φ is checked after the insertion of the tuple t in the network.

MoSL distinguishes between *path* and *state* formulae that are built from *basic state* formulae defined as follows:

$$\begin{array}{rl} \aleph ::= & A(p_1, ..., p_n)@\ell \rightarrow \Phi \quad | \quad !X@\ell \rightarrow \Phi \quad | \quad \langle T \rangle @\ell \rightarrow \Phi \\ | & A(p_1, ..., p_n)@\ell \leftarrow \Phi \quad | \quad \langle t \rangle @\ell \leftarrow \Phi \end{array}$$

The syntax of path formulae is the following:

$$\varphi ::= \quad \Phi\,{}_{\Delta}\mathcal{U}^{<t}_{\Omega}\,\Psi \quad | \quad \Phi\,{}_{\Delta}\mathcal{U}^{<t}\,\Psi.$$

The basic component of *path formulae* is the CTL *until* formula $\Phi\,\mathcal{U}\,\Psi$ parameterised with action sets: a path satisfies $\Phi\,{}_{\Delta}\mathcal{U}_{\Omega}\,\Psi$ whenever (eventually) a state satisfying Ψ (a Ψ-state) is reached via a Φ-path (i.e., a path composed only of Φ-states) and, in addition, while evolving between Φ-states, the performed actions satisfy Δ, and the Ψ-state is entered via an action satisfying Ω. Path formulae have also a time constraint. This is expressed by parameter t that requires that a Ψ-state be reached within t time units. Similarly, a path satisfies $\Phi\,{}_{\Delta}\mathcal{U}^{<t}\,\Psi$ if the initial state satisfies Ψ (at time 0) or a Ψ-state will be reached, within time t, via a Φ-path; in addition, while evolving between Φ-states only, actions satisfying Δ are performed.

The syntax of *state formulae* is the following:

$$\Phi, \Psi ::= \quad \mathsf{tt} \quad | \quad \aleph \quad | \quad \neg\Phi \quad | \quad \Phi \vee \Psi \quad | \quad \mathcal{P}_{\bowtie p}(\varphi) \quad | \quad \mathcal{S}_{\bowtie p}(\varphi)$$

They are divided in three groups. The first group includes formulae in propositional logic, where the atomic propositions are tt and the basic state formulae $\aleph$ introduced previously in this section. The second group includes statements about the likelihood of paths satisfying a property, $\mathcal{P}_{\bowtie p}(\varphi)$. Finally the third category includes formulae for the so-called long-run properties, $\mathcal{S}_{\bowtie p}(\varphi)$. State s satisfies the property $\mathcal{P}_{\bowtie p}(\varphi)$ if the total probability mass for all paths starting in s that satisfy φ meets bound $\bowtie p$. Here, $\bowtie$ is a binary comparison operator from the set $\{<, >, \leq, \geq\}$, and p a probability in $[0, 1]$. Long-run properties refer to the system when it has reached equilibrium. A state s satisfies $\mathcal{S}_{\bowtie p}(\varphi)$ if, when starting from s, the probability of reaching a state which satisfies Φ in the long run is $\bowtie p$.

SAM Verification of MoSL formulae against StoKlaim specifications is assisted by the SAM tool [16, 6], which uses a statistical model checking algorithm to estimate probabilities of property satisfaction by considering a set of independent observations. This algorithm is parameterised with respect to a given tolerance ϵ and error probability p, and guarantees that the difference between the computed values and the exact ones exceeds ϵ with a probability that is less than p. SAM is a command-line software tool developed in OCaML.

5 Formal Specification of SWS Compositions

We present in this section the Klaim formalisation of the reputation-based approach introduced in Section 3. In our specification we assume the presence of a given number of possible kinds of service compositions from which users can choose.

The overall SWSs system is rendered in Klaim by the following net:

$$s_{sws\_1} \ ::_{\rho^1_{sws}} P_{sws}(threshold_{sws\_1}) \ | \ \langle\text{``}swsType\text{''}, t_h\rangle \ | \ CompNetList_1 \ | \ CollNetList_1$$
$$\| \quad \ldots \quad \|$$
$$s_{sws\_m} \ ::_{\rho^m_{sws}} P_{sws}(threshold_{sws\_m}) \ | \ \langle\text{``}swsType\text{''}, t_k\rangle \ | \ CompNetList_m \ | \ CollNetList_m$$
$$\|$$
$$s_{comp\_net\_1} \ ::_{\rho^1_{comp}} \ SWS_{comp\_1} \quad \| \quad \ldots \quad \| \quad s_{comp\_net\_h} \ ::_{\rho^h_{comp}} \ SWS_{comp\_h}$$
$$\|$$
$$s_{coll\_net\_1} \ ::_{\rho^1_{coll}} \ SWS_{coll\_1} \quad \| \quad \ldots \quad \| \quad s_{coll\_net\_q} \ ::_{\rho^q_{coll}} \ SWS_{coll\_q}$$
$$\|$$
$$s_{user\_1} \ ::_{\rho^1_{user}} \ P_{user} \ | \ CompNetList_{user\_1} \ | \ \langle\text{``}composer\text{''}, P_1\rangle \ | \ \ldots \ | \ \langle\text{``}composer\text{''}, P_j\rangle$$
$$\| \quad \ldots \quad \|$$
$$s_{user\_n} \ ::_{\rho^n_{user}} \ P_{user} \ | \ CompNetList_{user\_n} \ | \ \langle\text{``}composer\text{''}, P_1\rangle \ | \ \ldots \ | \ \langle\text{``}composer\text{''}, P_k\rangle$$
$$\|$$
$$s_{rating\_server} \ ::_{\rho_{rs}} \ RatingList$$

where

$$\begin{aligned}
\rho^i_{sws} &= \{\mathbf{self} \mapsto s_{sws\_i}\} \\
\rho^j_{comp} &= \{\mathbf{self} \mapsto s_{comp\_net\_j}\} \\
\rho^k_{coll} &= \{\mathbf{self} \mapsto s_{coll\_net\_k}\} \\
\rho^t_{user} &= \{\mathbf{self} \mapsto s_{user\_t}, l_{rating\_server} \mapsto s_{rating\_server}\} \\
\rho_{rs} &= \{\mathbf{self} \mapsto s_{rating\_server}\}
\end{aligned}$$

Competition and collaboration networks, which represent the social aspects of the scenario, are expressed as collection of tuples organised in lists. In particular, a list x (e.g., *CompNet*) is specified in KLAIM as a collection of indexed tuples of this form:

$$xList \ \triangleq \ \langle\text{``}xList\text{''}, n\rangle \ | \ \langle\text{``}x\text{''}, 1, xItem_1\rangle \ | \ \ldots \ | \ \langle\text{``}x\text{''}, n, xItem_n\rangle$$

The tuple $\langle\text{``}xList\text{''}, n\rangle$, denoting that the list of type *xList* has length n, is used in KLAIM to read the whole list.

Each SWS and User in the system is rendered as a KLAIM node. Each User node has a list of known competition networks that are used to select the first SWS for new compositions (the first SWS is selected from the competition network with the highest reputation score). The node s_{rating_server} represents the central server collecting all the ratings given to the SWSs by users. The presence of this node is an abstraction of the rating storage system in use, that can be centralized or distributed. This abstraction does not affect the results of our analysis. Each SWS has a type and a list of competition and collaboration network to which it belongs. Each competition and collaboration network is rendered as a KLAIM node that contains the list of the SWSs belonging to the network.

Each SWS node runs the following KLAIM process, that describe SWSs' behaviour:

```
$P_{sws}(threshold) \triangleq$
    // wait for a composition request
    $\textbf{in}(\text{“}compositionReq\text{”}, !l_{requester}, !reqId)@\textbf{self};$
    $\textbf{in}(\text{“}numTransactions\text{”}, !counter)@\textbf{self};$
    // check the number of running transactions
    $\textbf{if}\ (counter \leq threshold)\ \textbf{then}\{$
        $\textbf{out}(\text{“}numTransactions\text{”}, counter + 1)@\textbf{self};$
        $\textbf{out}(\text{“}compositionResp\text{”}, \text{“}accept\text{”}, reqId)@l_{requester};$
        // a new transaction can be served
        $\textbf{eval}(P_{sws\_transaction}(reqId,\ l_{requester}))@\textbf{self}$
    $\}\ \textbf{else}\ \{$
        $\textbf{out}(\text{“}numTransactions\text{”}, counter)@\textbf{self};$
        $\textbf{out}(\text{“}compositionResp\text{”}, \text{“}reject\text{”}, reqId)@l_{requester}$
    $\};$
    $P_{sws}(threshold)$
```

Each SWS waits for a new composition request. Upon request receipt, the SWS checks if can fulfill it or not. The check is made on the basis of the number of ongoing transactions. A new service can be provided (via action $\textbf{eval}(P_{sws\_transaction}(reqId,\ l_{requester}))@\textbf{self}$) only if the check is positive.

The $P_{sws\_transaction}$ process is as follows:

```
$P_{sws\_transaction}(reqId,\ l_{requester}) \triangleq$
    // wait for a service request
    $\textbf{in}(\text{“}serviceReq\text{”}, l_{requester}, !data)@\textbf{self};$
    // the result depends on the QoS of the service
    $result := qos(data, s_{ws});$
    $\textbf{out}(\text{“}serviceQuality\text{”}, reqId, result)@l_{requester};$
    $\textbf{in}(\text{“}numTransactions\text{”}, !counter)@\textbf{self};$
    $\textbf{out}(\text{“}numTransactions\text{”}, counter - 1)@\textbf{self}$
```

A $P_{sws\_transaction}$ process is activated for each new composition and is used by SWSs to provide services. A transaction once activate waits for a service request. Once the request is received the result of the provided service, as well as its quality, is calculated by a function $qos()$ that we intentionally left unspecified. Once the service requested has been provided, the SWS closes the transaction.

Each user node runs the following KLAIM process:

```
$P_{user} \triangleq$
    // choose a composition
    $\textbf{in}(\text{“}composer\text{”}, !X_{composer})@\textbf{self};$
    // activate the chosen composition locally
    $\textbf{eval}(X_{composer})@\textbf{self};$
    // wait for the result of the composition
    $\textbf{in}(\text{“}finalResult\text{”}, compositionResult)@\textbf{self};$
    $rating := evaluateService(compositionResult);$
    $\textbf{out}(\text{“}evaluation\text{”}, \textbf{self}, rating)@l_{rating\_server};$
    $P_{user}$
```

Whenever a user wants to start a composition he (non-deterministically or probabilistically) chooses one of them, then it starts to compose services. Each service composition is managed by a composer. At the end of the composition, the user releases a rating referred to the quality of the whole composition of services. Such rating will be only used for evaluation purposes by users for future interaction with the SWSs.

A composer process is defined as follows:

```
$P_{composer\_i} \triangleq$
    // Search the first SWS of type '$t_x$' and send it a composition request
    $P_{select\_sws}(t_x);$
    $\textbf{in}(\text{“}mostTrusted\text{”}, !l_{trusted}, !repMT)@\textbf{self};$
    $resp = \text{“}reject\text{”};$
    $\textbf{while}\ (resp = \text{“}reject\text{”})$
        $id := freshId();$
        $\textbf{out}(\text{“}compositionReq\text{”}, \textbf{self}, id)@l_{trusted};$
        $\textbf{in}(\text{“}compositionResp\text{”}, !resp, id)@\textbf{self};$
        // Check the request response, if the SWS reject the composition
        //  search a new SWS in the competition network of $l_{trusted}$
        $\textbf{if}\ (resp = \text{“}reject\text{”})\ \textbf{then}\{$
            $P_{select\_comp\_sws}(l_{trusted});$
            $\textbf{in}(\text{“}mostTrusted\text{”}, !l_{trusted}, !repMT)@\textbf{self};$
        }
    }
    // When a SWS has accepted the composition request: send a service request
    $\textbf{out}(\text{“}serviceReq\text{”}, \textbf{self}, data)@l_{trusted};$
    $\textbf{in}(\text{“}serviceQuality\text{”}, id, result)@\textbf{self};$
```

```
// Evaluate the service and rate it
$rating := evaluateService(result);$
$\textbf{out}(\text{``}rating\text{''}, \textbf{self}, rating, l_{trusted})@l_{rating\_server};$
// Search the next SWS of type '$t_y$'
$P_{select\_coll\_sws}(t_y, l_{trusted});$
.........
// Check the request response, if the SWS reject the composition
//  search a new SWS in the competition network of $l_{trusted}$
.........
// Keep searching SWS till the composition is completed
```

The composer starts selecting the competition network with the highest reputation score. Inside this network it selects the SWS the highest reputation and it sends to the SWS a composition request. This is the first SWS of the composition. During the composition process, competition and collaboration networks are used to find SWSs. In particular, when a SWS rejects a composition a new SWS is selected from the competition network of the SWS who rejected the composition. Instead when a SWS accepts a composition the next SWS is selected from the collaboration network of the SWS who accepted the composition.

For each service received, the composer, in behalf of the user, gives a rating to the SWS. Such ratings are used to compute SWSs' reputation scores. Each composer $P_{composer\_i}$ has a different sequence of SWS to search, each composition has possibly a different number of SWS of different types.

The processes $P_{select\_sws}(type)$, $P_{select\_comp\_sws}(l_{sws})$ and $P_{select\_coll\_sws}(type, l_{sws})$ are used to implement the SWS selection strategies. A different strategy is used on the bases of when and where a SWS has to be selected: select the first SWS of the composition ($P_{select\_sws}(type)$), select a SWS after that a composition has been rejected ($P_{select\_comp\_sws}(l_{sws})$), select the next SWS of the composition ($P_{select\_coll\_sws}(type, l_{sws})$).

```
$P_{select\_sws}(type) \triangleq$
  // Initialization
  $\textbf{out}(\text{``}mostTrusted\text{''}, \textbf{self}, NO\_ONE)@\textbf{self}$
  $\textbf{in}(\text{``}netList\text{''}, !n)@\textbf{self};$
  $\textbf{for}\ j = 1\ \textbf{to}\ n\{$
      $\textbf{read}(\text{``}net\text{''}, j, !l_{net})@\textbf{self};$
      $\textbf{read}(\text{``}netType\text{''}, !net\_type)@l_{net};$
      // Check the type of sws in the network
      $\textbf{if}\ (net\_type = type)\ \textbf{then}\{$
          $P_{evaluate\_reputation}(l_{net});$
          $\textbf{in}(\text{``}reputation\text{''}, !rep)@\textbf{self};$
          $\textbf{in}(\text{``}mostTrusted\text{''}, !l_{trusted}, !repMT)@\textbf{self};$
```

```
            **if** $(rep > repMT)$ **then**{
                **out**("$mostTrusted$", $!l_{net}$, $!rep$)@**self**
            } **else** {
                **out**("$mostTrusted$", $!l_{trusted}$, $!repMT$)@**self**
            }
        }
    }
    **out**("$netList$", $n$)@**self**;
    // Select the competition network to use
    **in**("$mostTrusted$", $!l_{trusted\_net}$, $!repMT$)@**self**;
    **out**("$mostTrusted$", **self**, $NO\_ONE$)@**self**
    **in**("$swsList$", $!m$)@$l_{trusted\_net}$;
    // Search the most trusted SWS
    **for** $i = 1$ **to** $m${
        **read**("$sws$", $i$, $!l_{sws}$)@$l_{trusted\_net}$;
        // Calculate the SWS's reputation score
        $P_{evaluate\_reputation}(l_{sws})$;
        **in**("$reputation$", $!rep$)@**self**;
        **in**("$mostTrusted$", $!l_{trusted}$, $!repMT$)@**self**;
        **if** $(rep > repMT)$ **then**{
            **out**("$mostTrusted$", $!l_{sws}$, $!rep$)@**self**
        } **else** {
            **out**("$mostTrusted$", $!l_{trusted}$, $!repMT$)@**self**
        }
    }
    **out**("$swsList$", $m$)@$l_{net}$;
```

The $P_{select\_sws}(type)$ looks for the first SWS of the composition by searching in the competition networks it knows. It first selects the competition network, that contains the SWS of the desired type, with the highest reputation score. Inside the chosen network $P_{select\_sws}(type)$ selects the SWS with the highest reputation score.

The process $P_{evaluate\_reputation}()$ is used to compute the reputation score of a SWS, its specification depends on the type of reputation system in use. For our scenario we used the Beta reputation system [15].

```
$P_{select\_comp\_sws}(l_{sws}) \triangleq$
    // Initialization
    **out**("$mostTrusted$", **self**, $NO\_ONE$)@**self**
    **in**("$comNetList$", $!n$)@$l_{sws}$;
```

```
for $j = 1$ to $n${
    read(“$comNet$”, $j$, !$l_{net}$)@$l_{sws}$;
    in(“$swsList$”, !$m$)@$l_{net}$;
    for $i = 1$ to $m${
        read(“$sws$”, $i$, !$l_{new\_sws}$)@$l_{net}$;
        // Calculate the SWS’s reputation score
        $P_{evaluate\_reputation}(l_{new\_sws})$;
        in(“$reputation$”, !$rep$)@self;
        in(“$mostTrusted$”, !$l_{trusted}$, !$repMT$)@self;
        if ($rep > repMT$) then{
            out(“$mostTrusted$”, !$l_{new\_sws}$, !$rep$)@self
        } else {
            out(“$mostTrusted$”, !$l_{trusted}$, !$repMT$)@self
        }
    }
}
out(“$swsList$”, $m$)@$l_{net}$;
out(“$comNetList$”, $n$)@$l_{sws}$;
```

The $P_{select\_comp\_sws}(l_{sws})$ is invoked when a SWS in the composition rejected the request. Thus, this process select the new SWS from the competition network of the SWS that rejected. Inside this network $P_{select\_comp\_sws}(l_{sws})$ selects the SWS with the highest reputation score.

```
$P_{select\_coll\_sws}(type, l_{sws}) \triangleq$
    // Initialization
    out(“$mostTrusted$”, self, $NO\_ONE$)@self
    in(“$colNetList$”, !$n$)@$l_{sws}$;
    for $j = 1$ to $n${
        read(“$colNet$”, $j$, !$l_{net}$)@$l_{sws}$;
        in(“$swsList$”, !$m$)@$l_{net}$;
        for $i = 1$ to $m${
            read(“$sws$”, $i$, !$l_{new\_sws}$)@$l_{net}$;
            read(“$swsType$”, !$t$)@$l_{new\_sws}$;
            if ($t = type$) then{
                // Calculate the SWS’s reputation score
                $P_{evaluate\_reputation}(l_{new\_sws})$;
                in(“$reputation$”, !$rep$)@self;
                in(“$mostTrusted$”, !$l_{trusted}$, !$repMT$)@self;
```

```
                    if ($rep > repMT$) then{
                        out(“$mostTrusted$”, $!l_{new\_sws}$, $!rep$)@self
                    } else {
                        out(“$mostTrusted$”, $!l_{trusted}$, $!repMT$)@self
                    }
                }
            }
        }
        out(“$swsList$”, $m$)@$l_{net}$;
        out(“$colNetList$”, $n$)@$l_{sws}$;
```

Finally, the $P_{select\_coll\_sws}(type, l_{sws})$ is invoked when the next SWS of type $'type'$ has to be selected for an ongoing composition. In particular, given l_{sws} the locality of the last SWS in the ongoing composition. $P_{select\_coll\_sws}(type, l_{sws})$ selects from the collaboration network of l_{sws} the SWS with the highest reputation score.

6 Stochastic Analysis

In this section, we demonstrate how the Klaim specification presented in the previous section can support the analysis of reputation-based compositions of SWSs. In particular, in Section 6.1 and 6.2 we present results of simulation and model checking analysis, respectively.

Our approach relies on the formal tools presented in Section 4.2. Specifically, we enrich the Klaim specification introduced in the previous section with stochastic aspects (obtaining a StoKlaim specification). Then, the properties of interest (expressed in MoSL) are checked against the StoKlaim specification by means of the analysis tool SAM. As an excerpt of the StoKlaim specification, we report below the stochastic definition of process $P_{sws\_transaction}(reqId,\ l_{requester})$:

```
$P_{sws\_transaction}(reqId,\ l_{requester}) \triangleq$
    // wait for a service request
    in(“$serviceReq$”, $l_{requester}$, $!data$)@self : $\lambda_1$ ;
    // the result depends on the QoS of the service
    $result := qos(data, s_{ws})$;
    out(“$serviceQuality$”, $reqId, result$)@$l_{requester}$ : $\lambda_2$ ;
    in(“$numTransactions$”, $!counter$)@self : $\lambda_3$ ;
    out(“$numTransactions$”, $counter - 1$)@self : $\lambda_4$
```

The actions highlighted by a grey background are those annotated with rates λ_i.

6.1 Simulations

We present here the simulation results for the analysis of the reputation-based strategy for SWSs composition discussed in Section 3. We compare two strategies: reputation-based and random. In the random strategy, social networks are still used for SWSs selection, but SWSs selection is done randomly inside a network (i.e., once the network is chosen the selection of the next SWS is done randomly among the SWSs belonging to such network).

In our scenario, we consider the presence of: twelve SWSs of four different types, five users, four competition networks and two collaboration networks. Moreover, provided services can be rated as satisfactory or unsatisfactory by users, i.e. values in $\{0, 1\}$ are used to rate services. In the following we report the configuration of competition and collaboration networks, each competition network contains all the SWSs of a given type (e.g., the SWSs in competition network number one are of type one):

- Competition Network 1: SWS3, SWS6, SWS7;
- Competition Network 2: SWS1, SWS4, SWS8;
- Competition Network 3: SWS2, SWS5, SWS9;
- Competition Network 4: SWS10, SWS11, SWS12;

- Collaboration Network 1: SWS1, SWS2, SWS3, SWS5, SWS6, SWS12;
- Collaboration Network 2: SWS4, SWS7, SWS8, SWS9, SWS10, SWS11;

For our analysis, we assume SWSs' behaviours to be probabilistic and we model them by Bernoulli distributions with success probability θ. We used the Beta reputation system [15] to compute SWSs' reputation scores. The complete configuration is as follows:

SWS1: $\theta = 1$;	SWS2: $\theta = 0,2$;	SWS3: $\theta = 0,2$;
SWS4: $\theta = 0,2$;	SWS5: $\theta = 0,8$;	SWS6: $\theta = 0,8$;
SWS7: $\theta = 0,5$;	SWS8: $\theta = 0,2$;	SWS9: $\theta = 0,5$;
SWS10: $\theta = 0,2$;	SWS11: $\theta = 0,5$;	SWS12: $\theta = 0,8$;

We evaluate the quality of two possible services compositions. The first composition requires four services of type 4, type 2, type 3 and type 1. The second composition requires three different services, of type 3, type 1 and type 2. The quality of the composition is determined by the average of the ratings received for the single services of the composition. Figure 5 and 6 report the results of our analysis. From these charts we observe that a reputation-based strategy can improve the quality of the required composition. Indeed, when the user (or the composer) is driven by reputation scores for SWSs selection, the QoS of the requested composition is higher than in the case of random strategy. We believe that the random strategy can simulate a scenario in which the user (or the composer) has not additional information (excluding SNs) for SWSs selection.

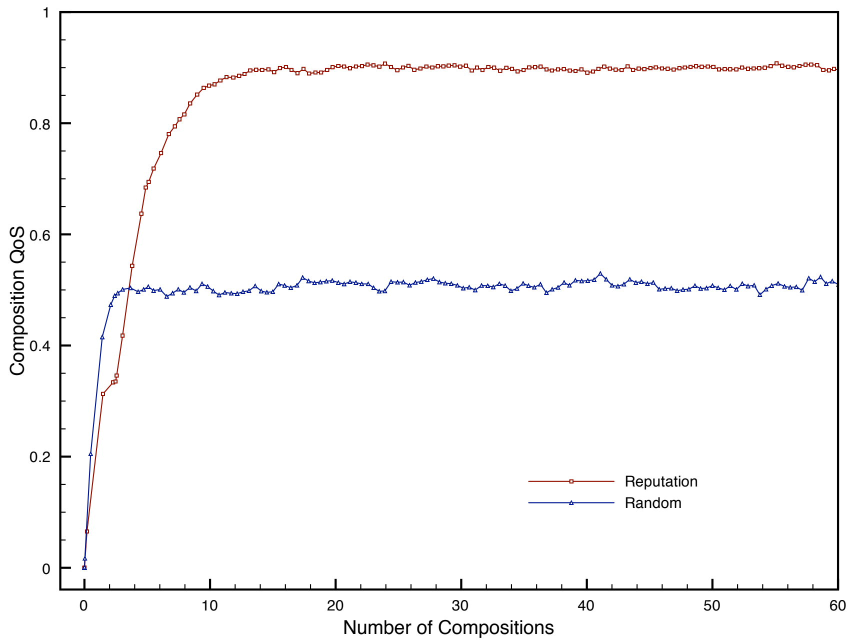

Fig. 5: Simulation results: composition 1

6.2 Model Checking

Here, we exploit model checking analysis to check properties of our reputation-based strategies. Specifically, we formalise a relevant property as a MoSL formula and verify it against the StoKlaim specification by means of the SAM tool. The property we investigate asks the probability with which *"a user gets a composition whose QoS is higher than a given threshold"*. This property is expressed in MoSL by the formula

$$\phi_{high\_QoS} = \langle \text{``}high\_qos\_composition\text{''} \rangle @\mathsf{ratings} \rightarrow true$$

This formula relies on the *consumption* operator $\langle T \rangle @ s \rightarrow \phi$, which is satisfied whenever a tuple matching template T is located at s and the remaining part of the system satisfies ϕ. Hence, the formula ϕ_{high_QoS} is satisfied if and only if a tuple $\langle \text{``}high\_qos\_composition\text{''} \rangle$ is stored in the ratings server tuplespace. Notice that the model of our system has been slightly modified to enable this analysis. In particular, a check on the QoS of a composition is done whenever it is completed and service tuple is produced on the ratings server whenever its QoS is higher then the threshold. Exploiting the previous formula, we can ask the probability with which *"a user gets a composition whose QoS is higher than a given threadshold within time t"*, defined as $true\ U^{\leq t} \phi_{high\_QoS}$, where the *until* formula $\phi_1 U^{\leq t} \phi_2$ is satisfied by all the runs that reach within t time units a state satisfying ϕ_2 while only traversing states that satisfy ϕ_1. The model checking analysis estimates the total probability of the set of runs satisfying such

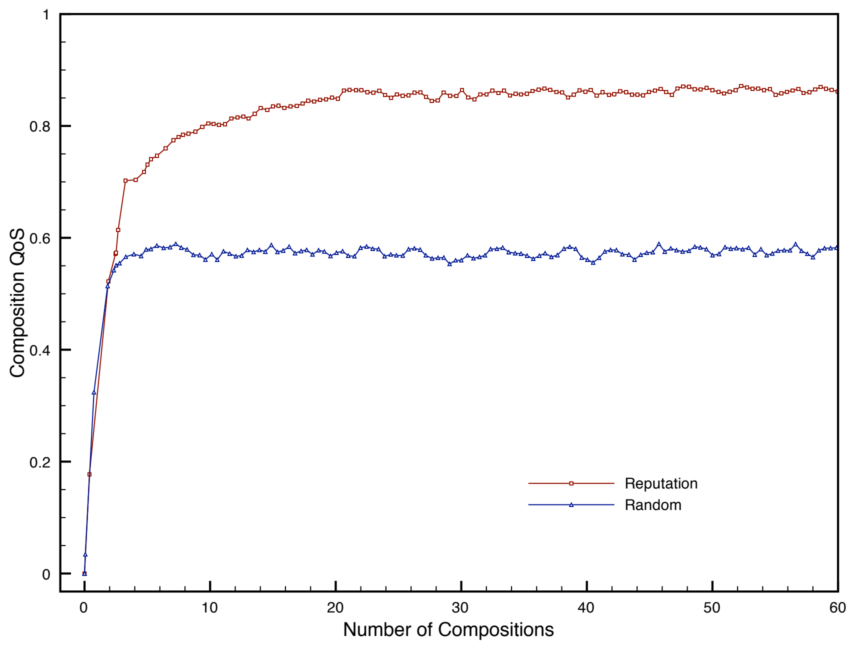

Fig. 6: Simulation results: composition 2

formula. The parameters ϵ and p of the model checking algorithm (see Section 4.2) have been both set to 0.05.

We verify the previous property for the two service compositions we discussed in Section 6.1. In particular, we fixed two different threshold values γ for each composition. Tables 2 and 3 report the results of the analysis for composition 1 and 2, respectively. In the tables are reported the result for different time interval t, specifically for t equals to 20, 30 and 40. For each value of t, we obtain a different expected number of service compositions. In particular, for the composition 1 the expected number of service compositions of this type for t equals to 20, 30 and 40 are 8, 12 and 17, respectively. Instead, for the composition 2 the expected number of service compositions of this type for t equals to 20, 30 and 40 are 9, 13 and 18, respectively.

The results show that, when a reputation-based strategy is used, the probability of getting a composition of high QoS is higher than in the case of a random strategy. Such results confirm what we already said in Section 6.1, giving us more details, e.g. about the probability of obtaining a composition whose QoS is higher than γ.

7 Concluding Remarks

In this paper, we made use of Klaim, a formal language specifically designed to model distributed systems, to formally specify a novel procedure for composition of Web services that belong to specialised social networks. The novelty in the way SWSs are

"a user gets a composition one whose QoS is higher than γ within time t"					
Selection Strategy	Probability	Selection Strategy	Probability	Time t	γ
Random	0.93524335	Reputation	0.97217869	20	0.6
Random	0.98912150	Reputation	0.99962605	30	0.6
Random	0.99860948	Reputation	0.99962605	40	0.6
Random	0.48456449	Reputation	0.80884996	20	0.9
Random	0.67872914	Reputation	0.97556725	30	0.9
Random	0.80071740	Reputation	0.99539034	40	0.9

Table 2: Model checking results: composition 1

"a user gets a composition two whose QoS is higher than γ within time t"					
Selection Strategy	Probability	Selection Strategy	Probability	Time t	γ
Random	0.99200178	Reputation	0.99522091	20	0.5
Random	0.99996490	Reputation	0.99996490	30	0.5
Random	0.74683925	Reputation	0.90101887	20	0.8
Random	0.89508888	Reputation	0.97810867	30	0.8
Random	0.95997986	Reputation	0.99369606	40	0.8

Table 3: Model checking results: composition 2

composed relies on considering their reputation. Reputation scores give a prediction of their QoS in future transactions, and make them more, or less, trustworthy to be part of a newly created composition. Supported by a set of KLAIM associated tools, we then analyse in a stochastic fashion properties of reputation-based SWSs composition. Results of the analysis show that reputation is a value-added attribute when composing SWSs. The synergetic combination of 1) being part of competition and collaboration social networks, and 2) capitalising on the outcomes of reputation systems, paves the way for automatic formation of high-quality SWSs compositions.

As future work, we envisage different directions. First, we aim at investigating different kinds of properties, as well as considering to rate services according to other attributes than QoS. Then, the KLAIM language could be extended with primitives that natively manage social aspects, while aiming at preserving the usage of the KLAIM tools.

References

1. J. Al-Sharawneh and M.-A. Williams. A Social Network Approach in Semantic Web Services Selection using Follow the Leader Behavior. In *Proceedings of the 13th Enterprise Distributed Object Computing Conference Workshops (EDOCW'2009)*, Auckland, New Zealand, 2009.
2. Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. Model-checking continuous-time markov chains. *ACM Transactions on Computational Logic (TOCL)*, 1(1):162–170, 2000.

3. Christel Baier, Joost-Pieter Katoen, and Holger Hermanns. Approximate symbolic model checking of continuous-time markov chains. In *CONCUR*, volume 1664 of *LNCS*, pages 146–161. Springer, 1999.
4. S. Bansal, A. Bansal, and M. B. Blake. Trust-based Dynamic Web Service Composition Using Social Network Analysis. In *Proceedings of the International Workshop on Business Applications for Social Network Analysis (BASNA'2010) held in conjunction with the Fourth International Conference on Internet Multimedia Systems Architecture and Applications (IMSAA'2010)*, Bangalore, India, 2010.
5. BizTechReports.Com. Web 2.0 strives to be collaboration software of choice for enterprises. In *http://goo.gl/d93W3K*, Last visited October 29, 2013.
6. Francesco Calzolai and Michele Loreti. Simulation and Analysis of Distributed Systems in Klaim. In *COORDINATION*, volume 6116 of *LNCS*, pages 122–136. Springer, 2010.
7. Alessandro Celestini, Rocco De Nicola, and Francesco Tiezzi. Specifying and analysing reputation systems with a coordination language. In *SAC*, pages 1363–1368. ACM, 2013.
8. W. Chen and I. Paik. Improving Efficiency of Service Discovery using Linked Data-based Service Publication. *Information Systems Frontiers, Spinger*, 2012 (forthcoming).
9. R. De Nicola, J.P. Katoen, D. Latella, M. Loreti, and M. Massink. Model checking mobile stochastic logic. *Theoretical Computer Science*, 382(1):42–70, 2007.
10. Rocco De Nicola, Gian Luigi Ferrari, and Rosario Pugliese. KLAIM: A Kernel Language for Agents Interaction and Mobility. *Trans. on Software Engineering*, 24(5):315–330, 1998.
11. Rocco De Nicola, Joost-Pieter Katoen, Diego Latella, Michele Loreti, and Mieke Massink. Model checking mobile stochastic logic. *Theor. Comput. Sci.*, 382(1):42–70, 2007.
12. Rocco De Nicola and Michele Loreti. Momo: A modal logic for reasoning about mobility. In *Formal Methods for Components and Objects*, pages 95–119. Springer, 2005.
13. E Allen Emerson and Edmund M Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer programming*, 2(3):241–266, 1982.
14. Edmond Gjondrekaj, Michele Loreti, Rosario Pugliese, Francesco Tiezzi, Carlo Pinciroli, Manuele Brambilla, Mauro Birattari, and Marco Dorigo. Towards a formal verification methodology for collective robotic systems. In *ICFEM*, volume 7635 of *LNCS*, pages 54–70. Springer, 2012.
15. Audun Jøsang and Roslan Ismail. The beta reputation system. In *Bled Conference on Electronic Commerce*, 2002.
16. Michele Loreti. SAM: Stochastic Analyser for Mobility, 2010. Available at `http://rap.dsi.unifi.it/SAM/`.
17. Z. Maamar, N. Faci, L. Krug Wives, H. Yahyaoui, and H. Hacid. Towards a Method for Engineering Social Web Services. In *Proceedings of the IFIP WG8.1 Working Conference on Method Engineering (ME'2011)*, Paris, France, 2011.
18. Z. Maamar, N. Faci, Q. Z. Sheng, and L. Yao. Towards a User-Centric Social Approach to Web Services Composition, Execution, and Monitoring. In *Proceedings of the 13th International Conference on Web Information System Engineering (WISE'2012)*, Paphos, Cyprus, 2012.
19. Z. Maamar, S. Kouadri Mostéfaoui, and H. Yahyaoui. Towards an Agent-based and Context-oriented Approach for Web Services Composition. *IEEE Transactions on Knowledge and Data Engineering*, 17(5), May 2005.
20. Z. Maamar, L. Krug Wives, Y. Badr, S. Elnaffar, K. Boukadi, and N. Faci. `LinkedWS`: A Novel Web Services Discovery Model Based on the Metaphor of "Social Networks". *Simulation Modelling Practice and Theory, Elsevier Science Publisher*, 19(10), 2011.
21. Z. Maamar, H. Yahyaoui, E. Lim, and P. Thiran. Social Engineering of Communities of Web Services. In *Proceedings of the 11th Annual International Symposium on Applications and the Internet (SAINT'2011)*, Munich, Germany, 2011.

22. Zakaria Maamar, Noura Faci, Alfred Loo, and Parisa Ghodous. Towards a quality of social network model in the context of social web services. In *IESS*, pages 297–310, 2012.
23. Zakaria Maamar, Noura Faci, Quan Z. Sheng, and Lina Yao. Towards a user-centric social approach to web services composition, execution, and monitoring. In *WISE*, pages 72–86, 2012.
24. A. Maaradji, H. Hacid, J. Daigremont, and N. Crespi. Towards a Social Network Based Approach for Services Composition. In *Proceedings of the 2010 IEEE International Conference on Communications (ICC'2010)*, Cap Town, South Africa, 2010.
25. M. Nam Ko, G. P. Cheek, M. Shehab, and R. Sandhu. Social-Networks Connect Services. *IEEE Computer*, 43(8), August 2010.
26. Q. Wu, A. Iyengar, R. Subramanian, I. Rouvellou, I. Silva-Lepe, and T. Mikalsen. Combining Quality of Service and Social Information for Ranking Services. In *Proceedings of ServiceWave 2009 Workshops held in conjunction with the 7th International Conference on Service Service-Oriented Computing (ICSOC'2009)*, Stockholm, Sweden, 2009.
27. X. Xie, B. Du, and Z. Zhang. Semantic Service Composition based on Social Network. In *Proceedings of the 17th International World Wide Web Conference (WWW'2008)*, Beijing, China, 2008.