International Journal of Innovative
Computing, Information and Control
Volume **10**, Number **1**, February **2014**

ICIC International ©2014 ISSN 1349-4198
pp. 115–132

A MARKET-BASED APPROACH TO THE DYNAMIC RECONFIGURATION PROBLEM OF SERVICE-BASED SYSTEMS

Romina Torres[1,2] and Hernan Astudillo[1]

[1]Departamento de Informática
Universidad Técnica Federico Santa María
Av. España 1680, Valparaíso, Chile
{ romina; hernan }@inf.utfsm.cl

[2]Facultad de Ingeniería
Universidad Andres Bello
Quillota 980, Viña del Mar, Chile
romina.torres@unab.cl

Received December 2012; revised April 2013

Abstract. *Modern complex distributed systems are built as service-based systems (SB Ss), composed of Web services, which are discovered and bound (most of the time) during runtime to handle the dynamic nature of Web service offerings. As far as we know, all reconfiguration approaches assign to SBSs' owners the responsibility to identify, select and compose services that satisfy their reconfiguration needs. In practice, SBSs are not omniscient regarding potential service providers' partners or offer trade-offs; thus, in most (or all) cases, service compositions to reconfigure SBSs are suboptimal. This article describes the market-based service reconfiguration (MBSR) approach, where service providers and requesters are software agents and reconfigurations naturally arise from the interaction of them. We implement MACOCO++, a multi-agent reference implementation to configure and reconfigure SBSs; the studied SBSs yield systematically better reconfigurations, and the prototype scales up well regarding both, complexity and number of concurrent service requests. Our results show that using an agent-based market metaphor yields better, fairer service agreements for requesters without needing a central (and possibly unfeasible) component with global knowledge.*

Keywords: Web services, Service-based systems, Service discovery, Service composition, Imperfect information, Multi-agent systems

1. **Introduction.** The service oriented architecture paradigm (SOA) drastically changed the way people make software from building systems from scratch to building them by composing distributed and heterogeneous pre-built services capable of fulfilling their functional and non-functional requirements (FRs and NFRs respectively) [1]. Web services (WS) [2] are one of the most popular techniques for building versatile distributed systems [3]. According to the W3C[1], *a Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format descriptor. Other systems interact with the Web service in a manner prescribed by its descriptor using messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards* [2]. A Web service is an abstract notion (declaring their functionalities) implemented by a concrete piece of software [2].

Service-based systems (SBSs) can be defined as flexible processes composed of abstract Web services [4], where typically the needed (abstract) services are bound to concrete ones

[1]http://www.w3.org/

at design and/or deploy time. Under a closed-world assumption [5] (where the boundary between systems and the environment is known and unchanging), these design-time service compositions are valid architectural configurations during runtime. Unfortunately, the dynamic nature of the services [5], where services become unavailable due to context changes, are withdrawn by their providers or drop their quality of service (by internal reasons or by being overloaded due to the growing demand of requesters) makes design-time configurations invalid during runtime requiring SBSs reconfigure their services frequently in order to keep satisfied the requirements. Dynamic reconfiguration is the process of making changes to an executing system without requiring that the system be temporarily shut down [6]. Typically, SBSs implement reconfiguration using (1) a requirements monitoring component that decides when a reconfiguration is needed and (2) a reconfiguration module capable of discovering, selecting and composing the proper services from a Universe of functionally-equivalent services only distinguishable by their non-functional properties (known NP-hard problems [7]). The frequency and how the reconfiguration process is applied (manual or automatic) depend on the internal policies of each system.

As far as we know, all the current approaches addressing the problem of obtaining architectural configuration replacements are request-driven. They delegate the responsibility to create configurations to requesters, who are not omniscient regarding the service providers' private information about offering trade-offs or partners affinity. Service offerings are rigid, there is no space for agreements negotiations, and then typically requesters are forced to accept the providers' conditions. Thus, in most (or all) cases, service compositions to reconfigure SBSs are suboptimal reconfigurations from the point of view of requesters.

In this article, we propose a market-based service reconfiguration (MBSR) approach that distributes the responsibility to obtain the reconfiguration replacements to service providers. The strategy of our proposal is to tackle the passive nature of the service providers during the discovery and composition processes. Services become active entities in a highly competitive and collaborative environment (the "service market"). They become aware of the SBSs' reconfigurations needs and organize themselves to propose reconfiguration alternatives. They compete with functionally-equivalent services in terms of their non-functional properties (e.g., price, response time, to name a few). They collaborate with other functionally-complementary services to create joint proposals. And, finally, they are willing to negotiate those negotiable aspects of their proposals with SBSs requesters, making from the point of view of requesters, agreed final proposals fairer than initial ones.

The remainder of this article is organized as follows: Section 2 presents the previous related work; Section 3 describes the MBSR approach; Section 4 describes *MACOCO++*, a prototypical implementation of MBSR using a multi-agent system; Section 5 describes simulation experiments conducted to validate our assertions; and Section 6 points out directions of future work, summarizes and concludes.

2. **Related Work.** The Web service architecture (WSA) [2] describes three possible implementations of the service discovery component: registry, index or peer-to-peer (P2P). In the registry and index implementations, service providers just publish their services' descriptors into registries (e.g., UDDI[2]), specialized open catalogues (e.g., Seekda[3]) or simply into the Web (e.g., Google[4]) and wait passively to be discovered by service requesters. In the P2P implementation, the central index or registry of previous implementations is

[2]UDDI: Universal Description, Discovery and Integration of Web services
[3]http://webservices.seekda.com/
[4]http://www.google.com

distributed between each service node. Then, each service provider help requesters on finding the proper service providers by broadcasting requests to all their service node neighbors. However, services are still passive entities of the discovery and composition processes, resulting in suboptimal configurations because requesters are not aware of the internal tradeoffs of the providers neither of the current affinity between them. Then, this approach allows to explore the solution space but they are incapable of exploiting it.

From the point of view of the adaptive systems [8] (where reconfiguration is performed each time there is enough evidence that the current configuration is violating the requirements), most (if not all) used techniques to obtain an architectural configuration are assuming that they have perfect information. Unfortunately, on the one hand, the current degree of change of the service market [5] makes almost impossible to maintain updated an all-known discovery service *Agency*, and on the other hand, always there is information that services cannot make public (e.g., their reserved price or their internal rules to manage their own businesses). To allow processes become flexible and adaptive, Ardagna and Pernici [4] proposed monitoring beforehand the services to be used at each process step in order to determine if there are better alternatives in the market (thus adding overhead to the system). The configuration or service composition problem was formalized by Ardagna and Pernici [4] as a mixed integer linear programming problem where optimization techniques are used to solve it. Unfortunately, compositions are obtained using a request-driven approach, where service providers are not involved (they are registry entries), and then they make decisions using incomplete information. The MOSES framework [9] supports QoS-driven runtime adaptation of service-oriented systems, which is needed when the QoS variability of services is not controlled by their providers, but can be affected by the variable demand that services are attending. They are gathering QoS information of services, but again services are not represented in this scheme and therefore, they are obtaining suboptimal configurations. Baresi and Guinea [10] proposed two languages to enrich BPEL processes with self-supervision capabilities, doting BPEL processes (a special kind of service-based systems) of monitoring and recovery strategies to adapt themselves during runtime to service violation agreements. However, again, services are just passive entries in registries, probably with incomplete or outdated information. The same problems can be found in the works of Filieri et al. [11] and Calinescu et al. [12]. The highly changing degree of the service market that was identified by Baresi et al. [5], makes almost impossible to maintain updated an “all-known” registry. There are several proposals from the biological-inspired algorithms area, where creating flexible and adaptive service-based systems is typically focused on the core service composition problem [13] (e.g., Ant Colony Optimization, Genetic Algorithm, Evolutionary Algorithms and Particle Swarm Optimization), but they have the same problem: they assume that they have updated and complete information about the services. Thus, obtain suboptimal alternatives.

Maamar et al. [14] published the first proposal where multi-agent was used to allow service compositions to arise from the inherent interaction of agents representing them. Different from our approach, they focus specially on deploying the service composition using agents. We instead focus on obtaining plausible architectural configurations (service composition) to implement resilient service-based distributed systems given technical specifications of software architects but without delegating the execution to agents. They do not address how to exploit the current solution space as our proposal that allows collaboration between service agents (e.g., creating a virtual organization) or negotiation (between service and requester agents, e.g., to obtain a fairer agreement for the requester). Tong et al. [15], have proposed a service agent model, which integrates web services, software agents and ontologies. First, assuming that is possible to have a common ontology

between services, they build a dependency graph, where the nodes are services and they are connected if and only if there is an output from one service that is the input of another. Second, each node is represented by an agent that can "speak" in the common language that customers use (using domains ontologies). Similar to our approach, when a customer specifies a request, an agent is created that broadcasts this request to all the provider agents. Then, each agent analyzes alternative paths over the graph. If they can consume the input, they try to find paths over the graph "cooperating" with other agents in order to produce the expected output. Unfortunately, the obtained service compositions or reconfiguration alternatives are suboptimal solutions from the point of view of requesters because they cannot negotiate with providers, the agreements conditions.

Norman et al. [16] and later, Kota et al. [17], have used virtual organizations to create and maintain service compositions against a dynamic, open and competitive environment. Norman et al. [16] applied this approach to mobile service provisioning, and Kota et al. [17] extended this last proposal adding it autonomic capabilities to any kind of agent organizations. Both approaches, similar to the previous discussed works [14, 15], use the multi-agent approach to determine the service composition, to run it, and to adapt it (if it is not fulfilling its objectives). The novelty of the proposal of Kota et al. [17] is that they transfer to the virtual organizations the capability to determine if they need to adapt themselves or not according to the failure records of their members. However, in these proposals there is no room for negotiation. It is important to notice that both, requesters and providers have for each non-functional property a desired and reserved value. Typically, there is an inverse relation between the desired and reserved values of providers and requesters (e.g., for the price property, on the one hand, requesters would like to pay the minimal price as possible, and on the other hand, providers would like to sell at the highest possible price). Anyway, when providers bid a request, they tend to propose for each requested property a value as closer as possible to their desired value. Thus, agreements obtained between providers and requesters tend to be suboptimal from the point of view of the requesters because the latter are forced to accept or reject but not to negotiate.

Zulkernine and Martin [18] proposed a trusted negotiation broker framework where a requester agent can negotiate with several providers using a specification model. According to the negotiation preferences of each agent, the negotiation broker selects the most appropriated negotiation strategy for each agent. During this process, agents can update the parameters of negotiation in order to reflect the current conditions of them (e.g., resource availability). However, this proposal does not ensure fairer agreements to all their participants, and it allows that some can take advantage from others. It is only applicable to service discovery process but typically SBSs' reconfigurations are service compositions.

3. **Towards a Market-Based Service Reconfiguration Approach.** In the classical service discovery model proposed in the WSA [2], on the one hand, service providers make available their service descriptions and, on the other hand, requesters retrieve Web service resources descriptions capable of implementing their requirements using a discovery service. However, in this model, service providers do not (and they should not) publish all the information needed about them to support requesters to discover a closer optimal set of services. For instance, which is the minimal price to accept a deal, which are the preferred providers to collaborate where they can obtain a lower cost, what are their tradeoffs between the different quality attributes, to name a few. If they would make public this private information, they could expose themselves to situations where others could take advantage from them. The service discovery model proposed in the WSA does not support providers to adapt their offerings according to each specific situation. Therefore,

service requesters (in our particular case SBSs) using the WSA are obtaining suboptimal reconfiguration alternatives because they are using only the incomplete information which is available in registries. Moreover, the information could be obsolete. Thus, the reconfiguration alternatives built by service requesters would be, potentially, better, if the available information would improve.

Our proposal is the market-based service reconfiguration (MBSR) approach, which transfers the responsibility to create (or maintain) these reconfigurations from the requesters to providers. We build a service market metaphor where service providers become aware of both, (1) the service reconfigurations needed by the SBSs and, (2) the service providers that are potential collaborators. Both, service requesters and providers, are wrapped as autonomous entities, whose goals are, to obtain (closer optimal) reconfigurations and to be selected, respectively. To this aim, we dote these entities of collaborative and competitive behaviors, in order for service providers to be able to both, create coalitions with complementaries providers and, relax their offerings when it is convenient (e.g., to be selected over their competitors or to guarantee fair agreements to requesters). We implement the service discovery model proposed in the WSA as a *Blackboard* component following a publisher-subscriber architecture: the *Agency*. The *Agency* has different topics (one for each functional capability provided by services). Providers subscribe their services into one or several topics according to their provided capabilities. Let PA, SA and RA be the autonomous entities representing the service provider, a specific service of a service provider, and a service requester respectively shown in Figures 1 and 2. It is important to notice that in this paper, we focus in the special case that SA are SBSs. Thus, Figures 1 and 2 show explicitly a SBS as a RA to exemplify our approach to support the dynamic reconfiguration of SBS using the MBSR approach.

The MBSR approach starts when a SBS (service requester) has detected that the current architectural configuration (service composition) no longer satisfies its requirements. As step (1) in Figure 1 shows, the SBS publishes a *call for tendering* indicating which functional categories and non-functional constraints (NFCs) are required ("the request")

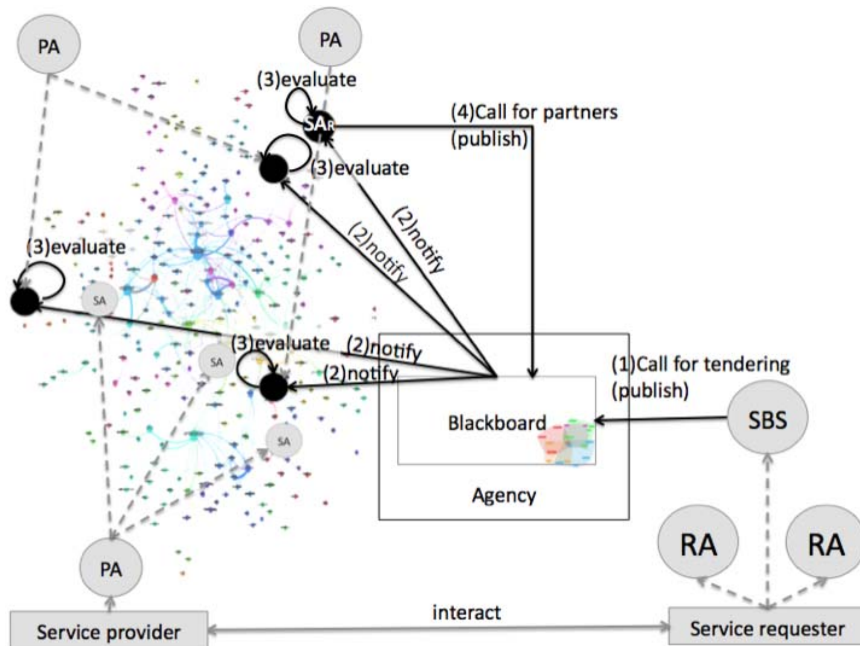
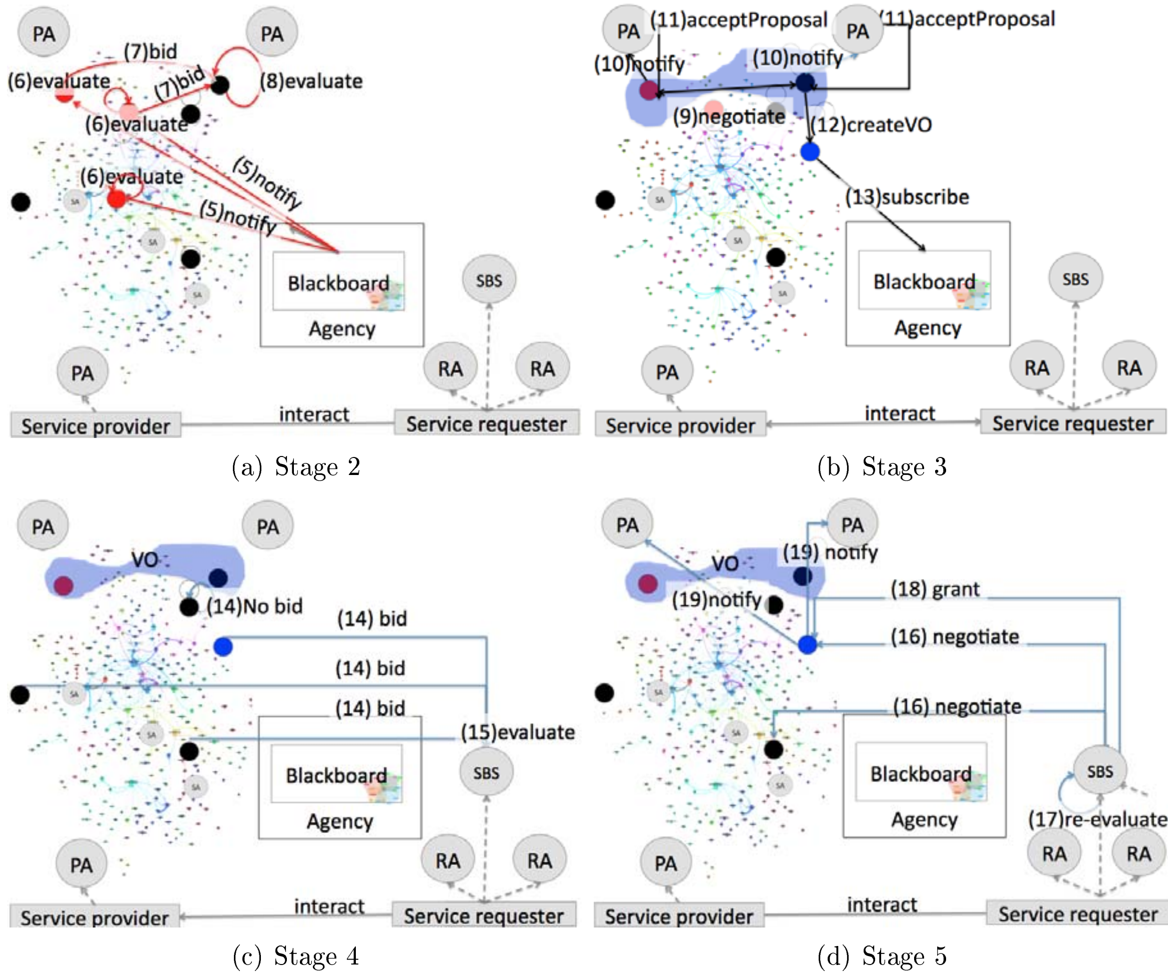


Figure 1. The initial stage of the MBSR approach: the service market organizing itself to create reconfiguration proposals to SBS requesters



(a) Stage 2 (b) Stage 3

(c) Stage 4 (d) Stage 5

FIGURE 2. The following stages of the MBSR approach: the service market organizing itself to create reconfiguration proposals to SBS requesters

into the *Blackboard*. Service providers subscribed to the specific topics are notified (step (2) *notify* in Figure 1). Different from the requester-driven approaches, in the MBSR approach services organize themselves to create the reconfigurations to satisfy SBSs' requests: (1) they analyze how well they satisfy them (step (3) *evaluate* in Figure 1), (2) they search for partners to create coalitions to satisfy complex requests (step (4) *call for partners* in Figure 1) and, also (3) they are willing to negotiate their proposals with the requesters (e.g., the price). When a provider calls for partners, it behaves like a requester, then we represent it in Figure 1 as SA_R. The *Blackboard* after receiving the request from SA_R, it sends notifications to those services subscribed to the functional topics demanded by SA_R (step (5) *notify* in Figure 2(a)). Each potential partner SA evaluates the new request (step (6) *evaluate* in Figure 2(a)) and then it decides if bid or not to the SA_R (step (7) *bid/no bid* in Figure 2(a)). Then, SA_R evaluates the coalitions' proposals (step (8) *evaluate* in Figure 2(a)) and negotiates coalitions' terms with their offerers (step (9) *negotiate* in Figure 2(b)). Each SA knows its PA; then during the negotiation each SA may ask to its PA to accept or reject coalition given the agreed conditions (steps (10) and (11) in Figure 2(b), *notify* and *accept/reject* respectively). SA_R re-evaluates and selects the partners creating a service coalition or a virtual organization VO [16] (step (12) *createVO* in Figure 2(b)). VOs are reusable, they are also autonomous entities, and their members can still perform independent operations. VOs subscribe themselves to

the *Blackboard* as any other service entity (step (13) *subscribe* in Figure 2(b)) encouraging their reuse by SBSs needing similar reconfigurations. A *RA* may receive several proposals from different *VO*s or a single *SA* entity (step (14) *bid/no bid* in Figure 2(c)), then it evaluates each proposal according to its own utility function (step (15) *evaluate* in Figure 2(c)) and it may decide to negotiate with some of them according to its private information and business rules given by its *PA* (step (16) *negotiate* in Figure 2(d)), re-evaluates the proposals (step (17) *re-evaluate* in Figure 2(d)) and it grants the contract (if it applies) to one of the offerers (typically to the higher re-ranked proposal) (step (18) *grant* in Figure 2(d)). Providers and requesters representatives (humans or code clients according to [2]) are notified of the agreed contract (step (19) *notify* in Figure 2(d)). Thus, the reconfiguration could be applied manually or automatically depending on the specific rules of each SBS.

4. **MACOCO++: A Multi-Agent Reference Implementation of the MBSR Approach.** In this section we present the multi-agent component composition framework, *MACOCO++*, which implements the MBSR approach by using a multi-agent system (MAS). A MAS is composed of multiple interacting agents (in this case we wrap *RA*s, *PA*s and *SA*s as software agents), where agents [19] are computer systems capable of both, autonomous action (decision making to achieve their goals) and interaction with other agents (cooperation, coordination and negotiation) [20].

In the following subsections we explain how a SBS build a reconfiguration request, how service agents organize themselves to propose alternatives, and how the SBSs can negotiate with proposals' providers to obtain fairer agreements. Figure 3 shows the main components of *MACOCO++* that will be used trough the different stages.

4.1. **Building the request.** In this subsection we explain how a SBS that is implementing their requirements (*Reqs*) (*FR*s as well as their *NFC*s) with an architectural configuration (*C*) obtains a reconfiguration from the service market (when appropriate). *MACOCO++* provides a functional taxonomy and nine non-functional constraints. Then, when a SBS needs from *MACOCO++* a reconfiguration, its requirements must be specified in such a way that *MACOCO++* "understands them". For instance, assuming that the *Reqs* of a specific SBS are the followings:

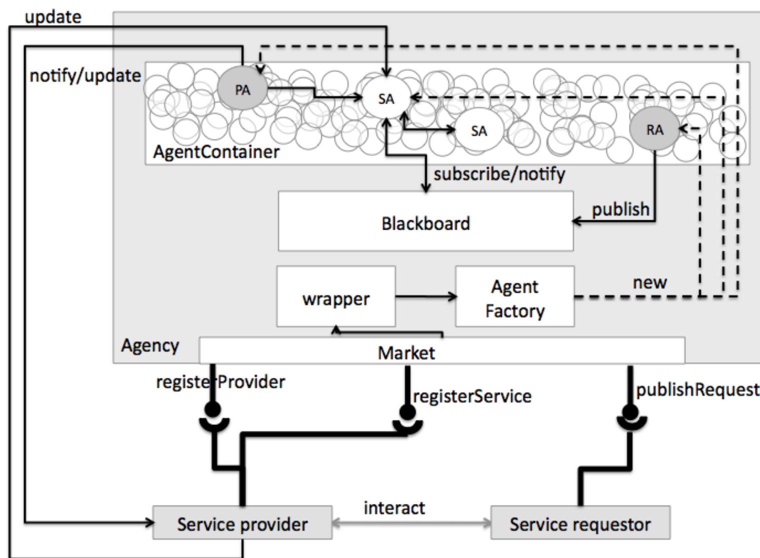


FIGURE 3. Main components of *MACOCO++*

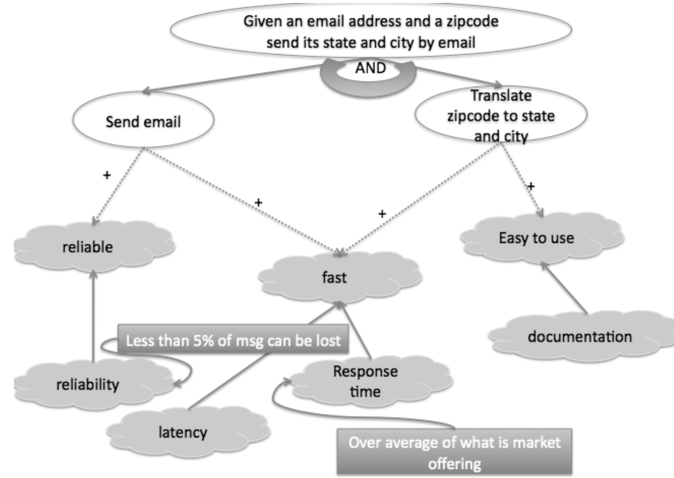


FIGURE 4. Goal-oriented analysis of *Reqs*

- Its FRs are *"When a user enters his/her email address and a zipcode, he/she expects to receive an email with the corresponding city and state to that code"*.
- Its NFRs are related with the performance of the system, and in this case "highly reliable" and "fast".
- And other constraints are that the system must be a SBS where services are easy to integrate for the developers.

The first task is to transform *Reqs* into S (the specification model) disambiguating the requirements by specifying them in terms of functional categories and numerical NFCs according to specific metrics.

Figure 4 shows the process of transformation from *Reqs* to S. The FRs of *Reqs* are divided into SR_1 and SR_2. By each one, NFCs are specified. Non-functional constraints must be verifiable, then they must be specified using a specific metric and a desired and a reserved numerical value. The desired value (D) represents the best case and the reserved value (R) represents the minimal acceptable case. Typically desired values are public but reserved ones are (and should be) not. Proposals with closer values to the desired values of the requesters will be higher ranked. Figure 4 shows the following:

- SR_1 is requiring a single or composite service capable to *send_email* with (1) a *response time* in the best case or D lesser than 150 and in the worst case or R at most 200 ms, (2) a *latency* with D lesser 5% and R at most 10%, and, (3) a *reliability* with D greater than 95% and R at least 80%.
- SR_2 is requiring a single or composite service capable to *return the city and state given a zip code* with (1) a *reliability* with D greater than a 95% and R not lesser than 80% and (2) a degree of how well documented is, of D at least a 95%, and R never lesser than 50%.

It is better if the services are free of charge ($D = 0$ dollars) but it is possible to pay as maximum 5 dollars by each one ($R = 5$). *MACOCO++* provides a user interface as Figure 5 shows, where is possible for SBSs' owners to select the specific functional categories codes corresponding to each functional sub-requirement (analogously with the NFCs). *MACOCO++* also allows to assign different degrees of importance to differentiate FRs as well as to differentiate NFCs. Figure 6 shows a XML representation of the example presented at the beginning of this section.

4.2. **Publishing the request.** As Figure 3 shows, *MACOCO++* exposes the endpoint *publishRequest* to SBSs in order that they can publish their reconfiguration needs. Assuming that SBSs' owners specify S in terms of functional categories and constraints known by *MACOCO++*, S is internally represented in *MACOCO++* as a multi-attribute utility function used to assess the satisfaction of the different configuration proposals received by self-organizing providers as follows:

$$eval(S(C)) : \sum_{i=1}^{I} v_i \left(\sum_{l=1}^{L} w^{[l,i]} \delta(c^{[l,i]}(C)) \right) \mathbb{I}(C, SR_i) \quad (1)$$

where $\{v_1, \ldots, v_i, \ldots, v_I\}$ is the set of relative importance of each software requirement SR_i; $\{w^{[1,i]}, \ldots, w^{[l,i]}, \ldots, w^{[L,i]}\}$ is the set of relative importance of each quality constraint l of the software requirement i; C is the reconfiguration under evaluation; $\mathbb{I}(C, SR_i)$ is the indicator function that returns 1 if there is a service $s \in C$ providing the functionality required to satisfy SR_i or 0 if not; $c^{[l,i]}(C)$ is a function that returns (if it applies) the current measurement value of the non-functional property l of the service $s \in C$ providing the functionality i; the function $\delta(c^{[l,i]})$ is a function which evaluates the degree of satisfaction of the current measurement (returned by the previous function) to the model S that depends on the R and D values provided by the architect. How $\delta(c^{[l,i]})$ is calculated depends on the type of property regarding if higher values are more desirable than lower ones. For instance, *reliability* is a "more is better" property and *price* is a"less is better". Algorithm 1 explains how $\delta(c^{[l,i]})$ is calculated in the two different cases.

As Figure 3 shows, when a requester, in this case SBS publishes a request in *MACOCO++*, the *Market* receiving this request asks to the *Wrapper* wraps the request into a requester agent (RA). The *Agent Factory* component creates the RA in the *AgentContainer*. Once the RA is in the *AgentContainer*, it publishes the request in all the relevant functional topics of the *Blackboard* component. Then, the *Blackboard* component notifies all the SAs that belong only to those functional topics. The steps 1 to 9 of Algorithm 2 summarize the publication of the request.

4.3. **Deciding what to offer and make a bid.** Steps 10 to 40 of Algorithm 2 summarize this subsection. Each, notified SA evaluates in which percentage satisfies the request, in

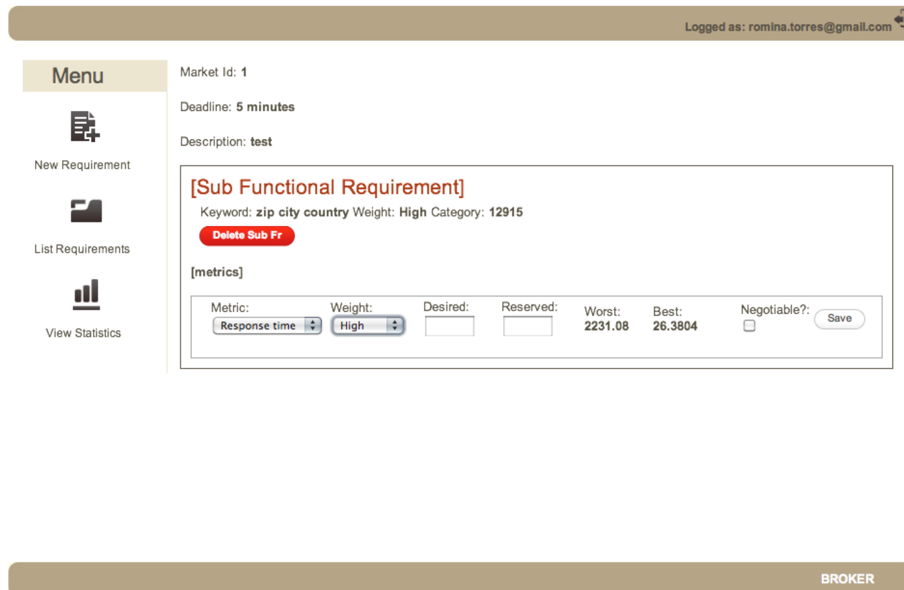


FIGURE 5. Web client to support requesters to build their requests

```xml
<?xml version="1.0" encoding="utf-8"?>
<request>
    <identification>
        <marketID>1</marketID>
        <clientId>2</clientId>
        <description>Description should be here ...</description>
    </identification>
    <deadline unit='minutes'>1</deadline>
    <callbackEndpoint>mailto:default/</callbackEndpoint>
    <fr>
        <subFr weight='H'>
            <keywords>send email</keywords>
            <nfr>
                <metric id='1' weight='H' negotiable='false' cut='1'>
                    <reservedValue>200</reservedValue>
                    <desiredValue>150</desiredValue>
                </metric>
                <metric id='4' weight='H' negotiable='false' cut='0'>
                    <reservedValue>80</reservedValue>
                    <desiredValue>95</desiredValue>
                </metric>
                <metric id='11' weight='H' negotiable='true' cut='0'>
                    <reservedValue>5</reservedValue>
                    <desiredValue>0</desiredValue>
                </metric>
            </nfr>
        </subFr>
        <subFr  weight='H'>
            <keywords>zipcode city state</keywords>
            <nfr>
                <metric id='4' weight='H' negotiable='false' cut='1'>
                    <reservedValue>80</reservedValue>
                    <desiredValue>95</desiredValue>
                </metric>
                <metric id='9' weight='H' negotiable='false' cut='0'>
                    <reservedValue>50</reservedValue>
                    <desiredValue>85</desiredValue>
                </metric>
                <metric id='11' weight='H' negotiable='true' cut='0'>
                    <reservedValue>5</reservedValue>
                    <desiredValue>0</desiredValue>
                </metric>
            </nfr>
        </subFr>
    </fr>
</request>
```


FIGURE 6. Example of a request

Algorithm 1 How the value of $\delta(c^{[l,i]})$ is calculated

1: **if** $R^{[l,i]} < D^{[l,i]}$, which means q_l is a "more is better" type of constraint **then**

2: $$\delta(c^{[l,i]}) = \begin{cases} 0 & \text{if } c_{[l,i]} < R^{[l,i]} \\ \frac{c^{[l,i]} - R^{[l,i]}}{D^{[l,i]} - R^{[l,i]}} & \text{if } R^{[l,i]} \leq c^{[l,i]} \leq D^{[l,i]} \\ 1 & \text{if } c^{[l,i]} > D^{[l,i]} \end{cases}$$

where $c^{[l,i]}$ is the current value provided by the potential candidate

3: **else**

4: $$\delta(c^{[l,i]}) = \begin{cases} 0 & \text{if } c^{[l,i]} > R^{[l,i]} \\ \frac{R^{[l,i]} - c^{[l,i]}}{R^{[l,i]} - D^{[l,i]}} & \text{if } D^{[l,i]} \leq c^{[l,i]} \leq R^{[l,i]} \\ 1 & \text{if } c^{[l,i]} < D^{[l,i]} \end{cases}$$

5: **end if**

order to decide to make a bid alone or not. If it satisfies the request, at least on A% of the required functionality goals, it can bid directly. If not, but it satisfies at least on a B% (where $B < A$), it may *call for partners* using the *Blackboard* component. By each not satisfied sub-request k, the SA is supported by K dedicated RA_k^{SA} (see steps 19 to 22 of Algorithm 2) to find the appropriated partners to fulfill those parts of the request that are not fulfilled by itself (where K is the number of subparts of the requests which are not supported by SA). Potential SAs of each sub part k of the request are notified in order that a SA is willing to create a coalition to build a collaborative proposal. The caller SA waits until its deadline is reached (where this deadline must be much lower than the global

Algorithm 2 Market-based service reconfiguration approach

Require: Given the *Blackboard* component with N functional topics.
Require: Given M services wrapped as service agents (SA) subscribed to topics of the *Blackboard*.
Require: Let t_X and t_Y be the maximum time that the service requestor and an internal supporting service requestor is willing to wait to get a solution respectively.
Require: Let A and B be parameters (with $A > B$) to regulate if SAs bid alone or not.

1: The service requestor publishes the $Request_X$
2: The Agent Factory wraps the $Request_X$ into the agent RA_X.
3: **for** $i = 1$ to $i = I$ (where $I \leq N$) **do**
4: RA_X publishes a call for tender at time T on the topic i of the *Blackboard*.
5: **for** $l = 1$ to $l = M_i$ (where M_i is the number of SA subscribed to the topic i) **do**
6: The *Blackboard* notifies each SA_l the $Request_X$
7: **end for**
8: **end for**
9: RA_X wait until $T + t_X$ is reached.
10: **for** $i = 1$ to $i = I$ **do**
11: **for** $l = 1$ to $l = M_i$ **do**
12: SA_l evaluates itself how much (sat_l) is satisfying the $Request_X$.
13: **if** $sat_l > A\%$ **then**
14: SA_l proposes an offer O as a single provider to RA_X
15: **else**
16: **if** $sat_l > B\%$ **then**
17: **if** SA_l decides to create a VO (it could decide not due to its internal state) **then**
18: The SA_l identifies the K subparts of the $Request_X$ which are not addressed by it.
19: **for** $k = 1$ to $k = K$ **do**
20: The *Agent Factory* creates a supporting service request agent $RA_k^{SA_l}$ to support SA_l on the selection for partners for the subpart k not addressed by it.
21: The $RA_k^{SA_l}$ publishes a call for tender in the respective (not addressed) topic of the *Blackboard* suspending itself until the deadline $T + t_Y$ is reached (where $t_Y = t_X * \frac{1}{K}$).
22: **end for**
23: **while** $currentTime < (T + t_Y)$ **do**
24: All supporting agents $RA_k^{SA_l}$ wait.
25: **end while**
26: **for** $k = 1$ to $k = K$ **do**
27: $RA_k^{SA_l}$ recovers from its queue the received offers by potential partners, evaluates them using the utility function presented in Equation (1) and selects the highest ranked.
28: $RA_k^{SA_l}$ notifies each selected SA.
29: **end for**
30: **end if**
31: **if** SA_l decides to make a bid as a VO (it could decide not due to its internal state) **then**
32: The *Agent Factory* creates VO^{SA_l} associating partners selected by RA^{SA_l}.
33: The VO^{SA_l} proposes an offer O as a VO to RA_X
34: **end if**
35: **end if**
36: **end if**
37: **end for**
38: **while** $currentTime < (T + t_X)$ **do**
39: RA_X waits
40: **end while**
41: RA_X recovers the offers from its queue and evaluates them using the function in Equation (1).
42: **if** RA_X has the negotiation feature activated **then**
43: See negotiation, in Subsection 4.4
44: **end if**
45: it creates a contract C_X between the RA_X and the providers of the higher ranked offer.
46: **end for**

deadline of the RA). Steps 26 to 29 of Algorithm 2 show how the partners are selected. After the RA reaches its deadline, it recovers the received offers (from single providers or VOs) and ranks them according to its utility function. If the agent is not wiling to negotiate, a contract is created between the RA and the providers of the selected offer (see step 45 of Algorithm 2). If not, next Subsection 4.4 explains how the negotiation is carried out by using a slightly different version of the Zeuthen strategy.

4.4. **Negotiating proposals to obtain for *RA*s fairer agreements.** After the RA received and evaluated the proposals, it can negotiate with SA or VO bidders those aspects which are negotiable. *MACOCO++* supports negotiation between requesters and offerers agents [21]. *MACOCO++* implements a negotiation strategy (slightly modified version of the Zeuthen strategy) that allows RAs to get fairer agreements. Typically, SAs propose offers which maximize their utility using their "desired" values for all the properties. Unfortunately for RAs, there is no mechanism to reverse this, letting RAs no choice but to accept or reject these offers. *MACOCO++* allows RAs to negotiate with the best n offerers by sending them (single SAs or VOs) a counteroffer proposing to modify the terms of the proposals (from the *desired* values of the offerers to values closer to the *desired* values of the requesters) for those aspects which are negotiable for both parts. To force that the negotiation starts, the RA sends its preferred deal (according to the Zeuthen strategy) with $risk = 1$ in order to force the offerer to concede (if it does not accept the deal in the first place). Algorithm 3 shows the setup of the negotiation process.

Algorithm 3 RA_X setting up the negotiation with the n best offers

Require: The n higher ranked offers before negotiation
1: **for** $i = 1$ to $i = n$ **do**
2: RA_X calculates its preference vector R_X such that $\text{Pr}_X \leftarrow \arg\max_{\text{Pr}} U_X(\text{Pr})$. At the beginning the vector Pr_X is set up with the desired values of the requester.
3: RA_X proposes as a counteroffer to offerer i Pr_X
4: **end for**

At each round of the negotiation process, the agent receiving the counteroffer (service or request agent) evaluates the utility that it would obtain by accepting the proposal of the counterpart. By simplicity, in this work we are assuming the utility function of RAs is exactly inverse to the utility function of SAs. For instance, requesters prefer to pay the lower possible price for a product while the providers prefer to sell it at the highest price.

We modified the original Zeuthen strategy in order to remove the imperative need of other agents to have to know private information of the other agents, in this case to know the utility function of the others. At each negotiation round, each agent calculates its own risk, and informs it to the counterpart; in this way, the other can compare and decide if it is its turn to concede or not. Each agent calculates its own utility with the offer of the counterpart, and also measures its risk. It compares the known risk of the counterpart and its own. The one with less risk should concede just enough so that it does not have to concede again in the next round. Then, the agent which concedes must determine how much it should concede. Instead of using the original mechanism of the Zeuthen strategy that demands that agents know the utility function of each other, we use a simple heuristic that basically adds 10% of the difference between *reserved* and *desired* values to all negotiable aspects (of course this parameters should be empirically determined). In this way the concession is almost always sufficient to invert the risks and force the other agent to concede on the next round. Then, the offers are sent and a new negotiation round starts again. If the concession is not enough to invert the risks, the other agent simply sends the same offer or the previous negotiation forcing the counterpart

Algorithm 4 Slightly modified version of the Zeuthen-monotonic-concession protocol

1: Agent j proposes Pr_j
2: **if** Agent j accepted or refused offer Pr_i **then**
3: Close negotiations and calculate $U_i(\mathrm{Pr}_i)$ if ACCEPT_PROPOSAL; otherwise stay with conflict deal.
4: **end if**
5: **if** $U_i(\mathrm{Pr}_j) \geq U_i(\mathrm{Pr}_i)$ **then**
6: Accept Pr_j, send ACCEPT_PROPOSAL. Close negotiations.
7: **end if**
8: $risk_i \leftarrow \frac{U_i(\mathrm{Pr}_i)-U_i(\mathrm{Pr}_j)}{U_i(\mathrm{Pr}_i)}$
9: **if** $risk_i \leq risk_j$ **then**
10: **if** all $c_i = R_i$ **then**
11: no new offer can be made, send REJECT_PROPOSAL. Close negotiations and stay with conflict deal.
12: **else**
13: calculate $\mathrm{Pr}_i \leftarrow \mathrm{Pr}'_i$ which concedes a little for each negotiable aspect moving a shorter step from *desired* to *reserved* values
14: **while** $c_i < R_i$ if $R_i > D_i$; or $c_i > R_i$ if $R_i < D_i$ **do**
15: add (or substract, depending the metric) 10% of the difference between R_i and D_i to new offer c'_i
16: **end while**
17: calculate new $risk_i(\mathrm{Pr}'_i)$
18: send new Pr'_i with $risk_i(\mathrm{Pr}'_i)$
19: **end if**
20: **else**
21: no need to concede, send same offer Pr_i
22: **end if**
23: wait for counteroffer or an ACCEPT_PROPOSAL or REJECT_PROPOSAL
24: **goto** Step 1

to re-evaluate and to propose a new offer. If no new offer can be made (meaning the agent reaches all its *reserved* values), it ends the negotiations with a conflict state. Algorithm 4 summarizes the negotiation procedure. The convergence of the Zeuthen strategy is guaranteed, as well as the final agreement is guaranteed to be individually rational and Pareto optimal. Even when in this work all agents are implementing the same negotiation strategy, agents could implement differently their negotiation behavior.

4.5. **Granting the contract.** After the negotiation ends, the RA grants the contract to the new higher ranked proposal (see step 45 of Algorithm 2). Then, RA sends ACCEPT_PROPOSAL and REJECT_PROPOSAL messages to the selected service or VO agents, which in turn notifies their PAs (using the *Notifier* component of *MACOCO++* not showed as main component in Figure 3).

4.6. **Dynamically reconfiguring SBSs.** SBSs have the responsibility of monitoring the compliance of the current C to their *Reqs*, in order to trigger a dynamic reconfiguration (replacing C by C') when appropriated. On those cases, SBSs can publish their *Reqs* to *MACOCO++* in order to get at design time an initial configuration and during runtime (by republishing) a proper replacement.

5. **Validation.** Previous version of MACOCO prototype was implemented over Netlogo 4.1[5], and further details can be found in [22]. Currently, MACOCO is the core component of the MBSR approach. It has been migrated from Netlogo to JADE[6]. Figure 3 shows

[5]http://ccl.northwestern.edu/netlogo
[6]http://jade.tilab.com/

the main components of *MACOCO++*. All the public functionalities of these modules are exposed as Web services. Over JADE, we have built the *Blackboard* component which is a publisher-subscriber component extending the current *yellow pages agent* of JADE. Although *MACOCO* has the capability to build a service market from an existent open catalog. We performed our experiments over a public and published dataset (QWS[7]), which has over 2500 WSDL-based Web services, each one with numerical measurements for nine QoS metrics. This dataset was generated in 2008; according to our latest certification tests, at 2012 only 1451 Web services from the original set are still operative, so we are using only that subset. All the components, including the Web client[8] (see Figure 5) to support service requesters on submitting valid requests were tailored to the structure of these services and metrics. The *Market* component was exclusively prepared to maintain this service market during runtime, where a script was built to register these services using the services of *MACOCO++*. In order to support negotiation and reconfiguration against a changing service market, service metadata was synthetically augmented and changes were introduced randomly.

All experiments presented in the following subsections were executed with a Dell Vostro 1400, Core 2 Duo T5470 (1.6Ghz, 2MB L2 Cache, 800MHz FSB), 2GB DDR2 667Mhz of RAM and 60GB 5400RPM SATA of hard drive.

5.1. **First experiment: from virtual organizations to Web service configurations.** This experiment tested the capacity of the service agents of *MACOCO++* to organize themselves to serve complex requests comprised by more than one sub-functional requirement. We measure the number of new VOs created as response to these requests, the reusing degree of current VOs when similar requests are published, and as usual, meantime and percentage of failed requests (request which were not satisfied). However, we start showing how the example introduced in Subsection 4.1 is solved with the MBSR approach.

Getting back to our example, we had a request comprised of two subparts, each one with one FR and multiple NFCs. These two sub-requirements are functionally classifiable in the functional topics *10012* ("send email") and *10344* ("return city and state given a zipcode"). Then, the *Agent Factory* creates the request agent RA_X, who publishes its request into the topics *10012* and *10344*. The subscribed SAs to these topics are notified. The $SA_{1000001} \in Topic_{10012}$ and $SA_{1000002} \in Topic_{10344}$, which were notified, evaluate separately the request, determining that each one only satisfies the request on a 50%. $SA_{1000001}$ decides to bid (according to its internal state); it starts a process to find partners in order to create a virtual organization that allows it to maximize the satisfaction of the current request. Because it asked to create a virtual organization, a new request agent RA_Y is created by the *Agent Factory.* The RA_Y publishes the sub-request in the $Topic_{10344}$. $SA_{1000002}$ subscribed to this topic, receives the notification and decides to bid because it fulfills 100% of the request. RA_Y waits until the deadline is reached and then recovers all the offers and ranks them according to its utility function. Assuming the highest ranked offer was received by $SA_{1000002}$, $SA_{1000001}$ decides to create a virtual organization with it, sending it an $ACCEPT_PROPOSAL$ message. Besides, it sends to the other offerers a $REJECT_PROPOSAL$ message. The VO is registered and wrapped immediately as a virtual organization agent (VO) that represents it to find similar requests which could be satisfied reusing the same services in the future. Then, the $SA_{1000001}$ desists to bid alone, empowering by the VO recently created to bid jointly the request. The agent RA_X reaches its deadline, recovers the offers and assuming the

[7] http://www.uoguelph.ca/~qmahmoud/qws
[8] http://dev.toeska.cl/broker

TABLE 1. Second experiment's results

id	minimal %	total time [min]	agents	#VOs	contract
R1	10%	10.171	43	20	yes
R2	10%	10.187	158	78	yes
R3	10%	10.409	67	65	yes
R4	40%	10.303	17	11	yes

highest ranked offer is the one received by the VO, it selects the VO and a contract is created between the request agent and the two component agents, members of the VO.

We used four requests with different number of functional sub-requirements as well as varying number of non-functional properties, which are indicated as metrics in the table: {1, *response time*}, {2, *availability*}, {3, *throughput*}, {4, *successability*}, {5, *reliability*}, {6, *compliance*}, {7, *best practices*}, {8, *latency*} and {9, *documentation*}. The possible values for weights are low, medium and high ({L, M, H} respectively). The test cases are the following:

- R1: {<13310,H,{<1,H>,<2,H>,<5,L>}><13392,H,{<4,H><9,M>,<7,M>}> <13400,H,{<2,M>,<9,M>,<8,M>}>}
- R2: {<13304,M,{<5,H>}><13302,H,{<3,H>}> <13395,L,{<2,L>}>}
- R3: {<13293,H,{<2,H>}><13294,L,{<2,L>}>}
- R4: {<13297,H,{<1,H>}> <13318,H,{<1,H>}> <13321,H,{<1,H>}> <13311,L,{<1,H>}> <13296,L,{<1,H>}>}

These test sets are executed twice: the first run allows to evaluate the system performance when agents generate virtual organizations to create reconfigurations, and the second run allows to compute the VO's reuse factor.

Table 1 shows the results. A service agent can bid if it satisfies at least a 10% (parameters $B = 10\%$ and $A = 100\%$) of the functional requirement in the first three requests and at least a 40% (parameters $B = 40\%$ and $A = 100\%$) in the last request. It is important to notice we are not varying the coverage percentage of the number of non-functional requirements. That means that for a service to satisfy the functional requirement's coverage, it must fulfill the 100% of the non-functional requirements. The time that an agent waits for bids, as well as the time an agent waits for partner proposals are set up to 10 minutes. Then, the total elapsed time cannot be less than 10 minutes; otherwise, there would be no time for the new VOs to make an offer. In the fourth column, the number of agents that try to create a coalition are shown. Due to the fact that the request $R1$ on the one hand, has more constraints for each subpart and, on the other hand, the number of subparts is three, it is more difficult for a single SA to fulfill it completely (moreover, in this experiment we set the constraints' coverage at 100%) and even to fulfill the minimal threshold ($B = 40\%$). Thus, only 43 SAs are asking to create coalitions because when the number of constraints is greater than one, the number of potential agents to serve it decreases. From the 43 coalition requests, only 20 VOs are created. The request R2, having the same number of sub requirements than $R1$ but a lower number of constraints, provokes in the service market that a greater number of agents try to satisfy it. At least there are 158 candidates that satisfy at least one subpart of the request (with at least the 100% of the constraints satisfied). From these 158 only 78 create VOs. R3 has only two subparts and the parameter. Only 67 decide to create VOs, where most of them conclude with the creation of the VOs. Finally, the request R4 has five subparts, then agents satisfying only partially subparts of this request, cannot bid alone ($A = 100\%$) and

TABLE 2. Re running second experiment

id	total time [min]	#VOs	contract
R1-rerun	0.384	20 VO	VO
R2-rerun	0.201	78 VO	VO
R3-rerun	0.441	65 VO	VO
R4-rerun	0.153	11 VO	VO

TABLE 3. Virtual organization members and their ranking for the four requests

id	subreq1	subreq2	subreq3	ranking
R1	122886434	136234335	90647582	1
R2	169192081	96268196		0.537037
R3	44881460	66815769		1
R4	99374147	166716168	101062406	1

neither can ask for partners ($B = 40\%$). Then, the number of agents capable of creating coalitions is reduced to 17, where only 11 creating a VO.

Table 2 shows the results of re-run for the second experiment. This time we have set up by default the deadline only in 10 minutes but the real elapsed time between the time when the request was published and the time when the offer of the winner VO arrives in the SA's offer queue was lesser than one minute. However, the deadline of 10 minutes was reached (as it was set up) and on the 100% of the requests the VO were selected again, obtaining a VO's reuse factor of 1. In Table 3 we present the ranking obtained by the virtual organizations that won the tendering by each request.

5.2. **Second experiment: negotiating the price.** We have synthetically added the price to the QWS dataset. We calculate a price which is directly proportional to the QoS. For each functional category and for each quality aspect we classified the services in five quality levels. Depending on which level the service belongs to, it gains points (between 1 and 5, where 1 is the worst level and five the best). Potentially, for each category, the best service could gain 45 points and the worst service could gain 9 points. Then, we calculate the corresponding price. In this experiment only the price aspect has been marked as a negotiable aspect.

The aim of this experiment is to assess how the negotiation process ensures, to SBSs to obtain fairer agreements. We set n to 1. Given the initial agreement, customers and providers negotiate the agreement using the modified Zeuthen strategy [21]. Table 4 shows the results. The first column indicates the id of the test case and the second the R and D price of the RA. The third column shows the initial and final utility of the RA and the fourth one shows the same for the SA. The last three columns show the overhead in seconds produced by the negotiation process, the number of rounds in which the negotiation converges to fair agreements and the variation of price from initial to final one. In seven of the ten cases the negotiation was successful. For instance in the test #1 the RA chose the offer of the SA offered to it to 42. Then, we forced the negotiation making the RA to send a counteroffer to the SA and after 7 rounds the utility of the RA increases a 14% with an overhead of almost 1 second. Of course, the utility of the provider SA drops from 1 to 0.85. The price which starts from the 42 drops to 39.3 which is not exactly the ideal price of the RA but was a fair agreement for both. Now, three of the ten cases the utility remained the same, then there was not negotiation, because the RA does not need to negotiate because it has what it asks.

TABLE 4. Overhead versus fairer agreements

test	$\Pr_{RA_{[R,D]}}$	$U_{RA_{I-F}}$	$U_{SA_{I-F}}$	**O** [s]	**#rounds**	$\Pr_{I-F}$
1	[46.2–38.4]	[0.8–0.94]	[1–0.85]	0.7	7	[42–39.3]
2	[50–42]	[0.91–0.91]	[1–0.98]	2.1	14	[40–39.2]
3	[83–69]	[0.82–0.82]	[1–0.975]	0.7	3	[60–58]
6	[78–65]	[0.97–0.98]	[1–0.98]	0.5	5	[66–64.6]
7	[71–59]	[0.98–0.98]	[1–0.967]	1.2	5	[56–54.4]
10	[47.5–39.6]	[0.81–0.9]	[1–0.9]	1.4	8	[44–42]

TABLE 5. Overhead produced by negotiation with several providers at the same time

#Counterparts	**Overhead** [s]	**#rounds with each offerer**	**Total of rounds**
1	0.78	$\{5\}$	5
3	1.47	$\{3, 5, 9\}$	17
5	3.4	$\{3, 4, 5, 5, 5\}$	22
10	4.2	$\{3, 3, 3, 4, 4, 4, 4, 5, 7, 8\}$	45

5.3. **Third experiment: overhead versus fairness.** The aim of this experiment is to show how the overhead increases as n (the number of offerers with which the client negotiates) increases (see Table 5). We perform this experiment with one request at a time.

The application scales. It is safe to encourage clients to allow their representing agents to negotiate with several offerers at the same time.

6. Conclusions.

This article addresses the dynamic reconfiguration problem of SBSs by using a distributed approach to obtain service compositions mitigating the problem of imperfect information of requester-driven approaches. The proposed MBSR approach extends the current Web services discovery model with active entities (agents representing service requesters and providers) and a virtual service market. MBSR agents are immersed in a multi-agent system, where potential configurations arise from the market, triggered by request agents that just publish a call for tendering on it and wait for bids. Bids emerge from service agents, which organize themselves by competing or collaborating to maximize the requests' satisfaction. Depending on the internal rules of agents, request agents can also negotiate with bidders to obtain fairer agreements to finally grant the contract (typically to the best offerers). Our approach avoid that SBSs are forced to accept offers just as providers dictate. We have used a negotiation strategy that allows to obtain fairer agreements to both (service providers and requesters but specially to requesters) as we proposed in [21].

By wrapping services and requesters as agents and reassigning the discovery responsibility from requesters to services we effectively could mitigate the imperfect information problem that requester-centered approaches experience at the moment of creating configurations. The MBSR approach allows services themselves to create closer optimal reconfigurations because services become aware of themselves, of others like them and of what requesters need.

Acknowledgment. This work is partially supported by projects VirtualMarket (Fondef CA12i10380), UTFSM-DGIP 24.12.50, and CCTVal (Basal FB0821).

REFERENCES

[1] M. P. Papazoglou, P. Traverso, S. Dustdar and F. Leymann, Service-oriented computing: State of the art and research challenges, *Computer*, vol.40, no.11, pp.38-45, 2007.

[2] D. Booth, H. Haas and F. McCabe, Web services architecture, World Wide Web consortium, *Tech. Rep.*, http://www.w3.org/TR/ws-arch/, 2004.

[3] Z. Zheng, Y. Zhang and M. R. Lyu, Distributed QoS evaluation for real-world Web services, *IEEE International Conference on Web Services*, pp.83-90, 2010.

[4] D. Ardagna and B. Pernici, Adaptive service composition in flexible processes, *IEEE Trans. on Softw. Eng.*, vol.33, no.6, pp.369-384, 2007.

[5] L. Baresi, E. D. Nitto and C. Ghezzi, Toward open-world software: Issue and challenges, *Computer*, vol.39, no.10, pp.36-43, 2006.

[6] J. Hillman and I. Warren, An open framework for dynamic reconfiguration, *Proc. of the 26th International Conference on Software Engineering*, pp.594-603, 2004.

[7] C. Zhang, R. N. Chang, C.-S. Perng, E. So, C. Tang and T. Tao, QoS-aware optimization of composite-service fulfillment policy, *IEEE SCC*, pp.11-19, 2007.

[8] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum and A. L. Wolf, An architecture-based approach to self-adaptive software, *IEEE Intelligent Systems*, vol.14, no.3, pp.54-62, 1999.

[9] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. L. Presti and R. Mirandola, MOSES: A framework for QoS driven runtime adaptation of service-oriented systems, *IEEE Trans. on Software Eng.*, vol.38, no.5, pp.1138-1159, 2012.

[10] L. Baresi and S. Guinea, Self-supervising bpel processes, *IEEE Trans. on Software Eng.*, vol.37, no.2, pp.247-263, 2011.

[11] A. Filieri, C. Ghezzi and G. Tamburrelli, A formal approach to adaptive software: Continuous assurance of non-functional requirements, *Formal Aspects of Computing*, vol.24, pp.163-186, 2012.

[12] R. Calinescu, C. Ghezzi, M. Kwiatkowska and R. Mirandola, Self-adaptive software needs quantitative verification at runtime, *Commun. ACM*, vol.55, no.9, pp.69-77, 2012.

[13] L. Wang, J. Shen and J. Yong, A survey on bio-inspired algorithms for web service composition, *CSCWD*, pp.569-574, 2012.

[14] Z. Maamar, S. K. Mostefaoui and H. Yahyaoui, Toward an agent-based and context-oriented approach for web services composition, *IEEE Trans. on Knowl. and Data Eng.*, vol.17, no.5, pp.686-697, 2005.

[15] H. Tong, J. Cao, S. Zhang and M. Li, A distributed algorithm for web service composition based on service agent model, *IEEE Trans. on Parallel Distrib. Syst.*, vol.22, no.12, pp.2008-2021, 2011.

[16] T. J. Norman, A. Preece, S. Chalmers, N. R. Jennings, M. Luck, V. D. Dang, T. D. Nguyen, V. Deora, J. Shao, W. A. Gray and N. J. Fiddian, Agent-based formation of virtual organisations, *Knowledge-Based Systems*, vol.17, no.2-4, pp.103-111, 2004.

[17] R. Kota, N. Gibbins and N. R. Jennings, Decentralized approaches for self-adaptation in agent organizations, *ACM Trans. on Auton. Adapt. Syst.*, vol.7, no.1, pp.1:1-1:28, 2012.

[18] F. H. Zulkernine and P. Martin, An adaptive and intelligent SLA negotiation system for web services, *IEEE Trans. on Services Computing*, vol.4, pp.31-43, 2011.

[19] M. Wooldridge and N. R. Jennings, Intelligent agents: Theory and practice, *Knowledge Engineering Review*, vol.10, no.2, pp.115-152, 1995.

[20] M. Wooldridge, *An Introduction to Multiagent Systems*, John Wiley & Sons, Inc., New York, NY, USA, 2001.

[21] R. Torres, D. Rivera and H. Astudillo, Web service compositions which emerge from virtual organizations with fair agreements, *Proc. of the 6th KES International Conference on Agent and Multi-Agent Systems: Technologies and Applications*, pp.34-43, 2012.

[22] R. Torres, H. Astudillo and E. Canessa, MACOCO: A discoverable component composition framework using a multiagent system, *International Conference of the Chilean Computer Science Society*, pp.152-160, 2010.