# Market-awareness in Service-based Systems

Romina Torres
*Universidad Tecnica Federico Santa Maria*
*Chile*
*INRIA Paris-Rocquencourt, France*
*romina@inf.utfsm.cl*

Nelly Bencomo
*INRIA Paris-Rocquencourt*
*France*
*nelly@acm.org*

Hernan Astudillo
*Universidad Tecnica Federico Santa Maria*
*Chile*
*hernan@acm.org*

***Abstract*—Service-based systems are service consumer applications built by composing pre-existing services. Both, service providers and consumers are on a service market that is constantly changing. We see different problems in the way service-based systems are currently carried out. On the one hand, consumers define requested services using specifications that depend on the current knowledge of the market. Such specifications may become obsolete due to the continued improvements in the QoS of the services in the market. On the other hand, service providers are passive entities waiting to be discovered by the service consumers.**

**In this work, we propose a framework to support both, consumers and service providers to be aware of the changes in the market. With our approach consumers are able to specify the required QoS using abstract specifications that will be eventually concretized at runtime and according to real characteristics of the market. Services are represented by active software agents. These agents are collectively aware of themselves and what the market requires. Agents are able of creating and maintaining virtual organizations that not just wait to be discovered but also react actively to demands of the market providing self-adaptation capabilities to service-based systems. Explanations of the current prototype are provided.**

***Keywords*-multi-agent systems; virtual organizations; self-adaptive systems; service-based systems; uncertainty; awareness; service market; model@runtime**

## I. Introduction

Requirements-aware systems [1] have addressed the need of reasoning about uncertainty to support runtime adaptations. Systems that are aware of their requirements can deal with different kinds of uncertainty [2] by monitoring their requirement satisfaction at runtime to execute corrective actions when deviations are detected (e.g triggering adaptations). One the uncertainties that have not been studied is the uncertainty related to changes in the service market.

Traditional software development were based on the closed-world assumption that the boundary between system and environment is known and unchanging [3]. For Service-based systems (SBSs), this assumption cannot be longer maintained due to the unpredictable and uncertainty in the behavior of the service market ( improvements in qualities of services, services becoming available or leaving the market to name a few).

In this paper we identify two dynamics produced by the constantly changing service market. As any other market, supply/demand drives the evolution of the service market. On the one hand, in order to gain niche markets, functionally-equivalent services are constantly competing (typically in terms of QoS and price). On the other hand, the supply of service shapes the specifications of requests done by client systems. To model the above dynamism, we propose modeling service markets as self-organizing systems (implemented as multi-agent systems) where service providers and consumers are represented by agents that negotiate on their behalf. We make both, service suppliers and consumers, aware of the market: (1) service providers are collectively aware of the current demands of the market, other competitors and even potential partner to provide services in conjunction and, (2) service consumers specify required QoS using abstract specifications that will be eventually concretized at runtime and according to real characteristics of the market.

The rest of the article is organized as follows: Section II introduces our approach to mitigate the obsolescence of the quality-specification models in service-based systems; Section III introduces our approach to model the service market; Section IV explains our proposal to present a solution, which support service-based systems and services themselves to address the uncertainty of the market at runtime; Section V highlights our current implementation and Section VI concludes the paper and presents our ongoing work.

## II. Mitigating the obsolescence of quality-specification models in SBS

During the specification of a system, the requirements $R$ are transformed into specification $S$ supported by relevant domain knowledge $K$ [4]. According to Zave and Jackson [4], the specifications $S$ and the relevant domain knowledge $K$ must be sufficient to guarantee that the requirements $R$ are satisfied:

$$S, K \vdash R \tag{1}$$

During execution, it is possible to determine if the requirements $R$ are satisfied or not, by monitoring the deviations between the system's behavior and the specification models $S$ [4]. Crucially, the latter is valid only if $K$ has not considerably changed during execution since the specification $S$ were defined. For the specific case of SBS, this

assumption cannot always be guaranteed [3]. Even, if the required functionalities of a SBS does not change, the quality specifications which constrain functionalities are likely to change quickly because they are highly dependent on the characteristics of the market represented by $K$. In this kind of systems, the quantifiable quality specifications $S$ are obtained by observation of what the service market $K$ is offering. Unfortunately, during execution the ever changing market may provoke the obsolescence of $S$ making impossible for systems to determine if their requirements $R$ are being satisfied or not by the current configuration of services $C$ used. The result is that SBSs could miss opportunities of adaptations because they are not aware when requirements are becoming unsatisfied.

In [5], we have proposed an approach to tackle the above. We addressed the uncertainty associated to QoS of services due to the unforeseen behaviour of the market during runtime.

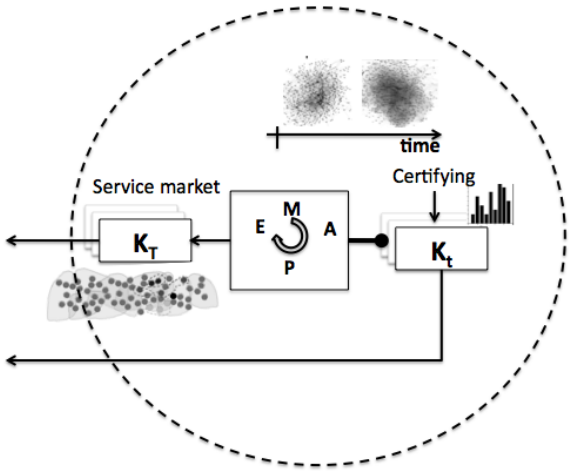


Figure 1. serviceMarket

Let $CS_i = \{s_i^1, s_i^2, ..., s_i^{n_i}\}$ be a functionally-equivalent service set, which is comprised by $n_i \geq 1$ concrete services that provide the same functionality $i$ than an abstract service $sa_i$, with $1 \leq i \leq I$ [6]. Let $Q = \{q_1, \ldots, q_M\}$ be the set of quality attributes which allow us to distinguish functionally-equivalent services. Let $K$ be the service market composed by all the functionally-equivalent service sets. We assume that for each functionally-equivalent services set $CS_i$, it is possible to certify services by periodically obtaining measurements for each quality attribute of each service member (illustrated in Figure 1 as $K_t$). Services belonging to each $CS_i$ can be ordered depending on each of their quality attributes. According to these histograms, it is possible to determine $c$ overlapping groups. Each group is a linguistic variable $LV^{[}i]$. For the sake of simplicity, we use the same five variables for each quality attribute $LV_j^{[i]} = \{$"poor", "fair", "good", "very good", and "excellent"$\}$. Each linguistic variable is a fuzzy set denoted by $\mu$ with a triangular shape (more details can be found in [5]).

$K_T$ allows clients to specify its quality specifications by using the $LV$s, while $K_t$ allows the systems to monitor if the current architectural configuration $C$ is satisfying the concrete specifications $S$. The information which can be obtained from $K_T$ and $K_t$ is totally different. For instance from $K_T$ we can obtain *the services with "excellent" response time of the functionally-equivalent set $CS_i$ are those which response time value is less than 43 milliseconds* and from $K_t$ we can obtain *the service with id* 23243466 *which belongs to the set $CS_i$ at time t has a response time of* 113 *milliseconds*.

Figure 1 shows that we are constantly taking snapshots from the current QoS of the services, $K_t$, as well as monitoring if we have gathered enough evidence that the market is changing to recalculate the meaning of $LV$. Notice the frequency of generate $K_T$ is, of course, significantly lower than the frequency of take the snapshots $K_t$. We define the abstract language $\text{Ł}(K_T)$ as the set of variable $LV$ and their numerical meaning at runtime to be used by consumers to annotate the requirements $R$, transforming them in abstract specifications $S^*$ or market-aware models. We encourage clients to specify their requirements using linguistic variables instead of precise numbers because the latter are more prone to obsolescence. Using our approach, the real numerical meaning of each linguistic variable are delayed until the moment they are used.

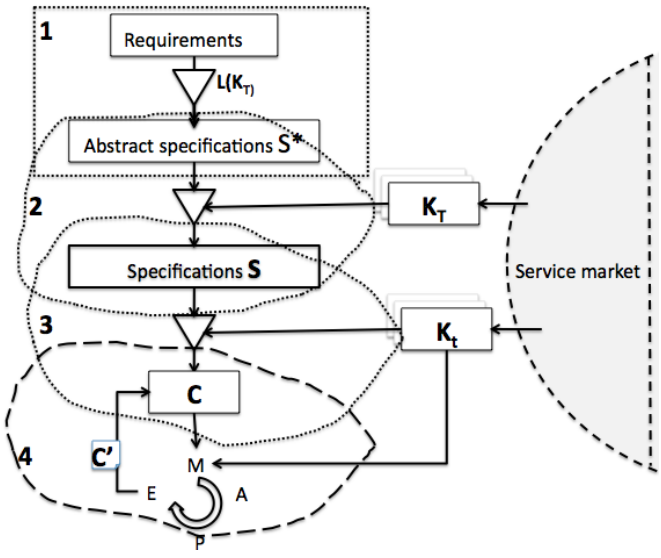


Figure 2. From market-aware requirements to self-adaptive systems at runtime

Figure 2 shows an overview of the main subprocesses of our approach to support clients to create a market-aware specification model at design time which will be aware of the market changes at runtime. The subprocesses are explained

as follows:

- **Subprocess 1: From Requirements to Abstract Specifications**. Using the abstract language $Ł(K_T)$ composed of linguistic variables $LV$ we transform requirements $R$ into abstract specifications $S^*$. $S^*$ is constructing as fuzzy conditional statements (e.g. *IF the response time of a service capable of sending email is at least "fast" THEN the belonging degree to the acceptable solution set is high*) which are represented as a fuzzy multicriteria decision making function which must be satisfied $\sum_{i=1}^{I} v_i \left( \sum_{j=1}^{J} w_j^{[i]} \delta_{MAC_j^{[i]}}(c_j(C)) \right) \mathbb{I}(C, SR_i)$ (deeper details can be found in [5]). This subprocess must be executed each time the requirements $R$ changes.
- **Subprocess 2: From Abstract Specifications to Concrete Specifications**. Using the abstract specifications $S^*$ and the current knowledge domain $K_T$, the abstract specifications $S^*$ are mapped into concrete specifications. Each time the market has changed $K_T$, which is reflected in the numerical meaning of $LV$, this subprocess must be re-executed.
- **Subprocess 3: From Concrete Specifications to architectural configuration**. Using the concrete specifications $S$ and the current snapshot of the market view $K_t$ the architectural configurations $C$ that better satisfy the model can be determined (deeper details can be found in [5]).
- **Subprocess 4: Adapting architectural configuration at runtime**. Each time a $C$ is generated, it must be subscribed to monitoring in order to allow to the system to check if at runtime $C$ still satisfies $S$. If services $s \in C$ do not satisfy $R$ at time $t$ and according to the Analyzer component, this fault is not transient but recurrent, then a new configuration $C'$ should be calculated to replace the current one.

## III. From virtual organizations to service compositions

In [7] we pointed out that, historically, the responsibility to discover the proper services to satisfy the market demand has been delegated to services consumers who need to divide themselves their requests, to search services, for each part to evaluate different alternatives, and to construct their solutions by building compositions of these alternatives. Up to now, current service composition techniques have modeled services as passive entities waiting to be discovered by service consumers. In this section we present MACOCO+ which implements service providers as active software agents collectively aware of other services. MACOCO+ provides a blackboard where all the demands from service consumers are published, making services also aware of the market.

We proposed MACOCO (Multi-Agent COmponent COmposition) [7], an innovative approach, whose first version allows clients to discover services, which were subscribed to the blackboard core component. Both, services consumers and providers were represented as software agents. Both, were made aware of their environment (the market) by subscribing them to the blackboard. MACOCO was inspired on the CONOISE-G platform [8], which allows to create and maintain virtual organizations to satisfy complex requests for e-commerce. As CONOISE-G, the second version of MACOCO [9], allows to service agents to communicate to each other to create virtual organizations (VOs) which satisfy complex requests. Then, the service compositions naturally raised from the market. The current version of MACOCO, MACOCO+ [10], also allows agents negotiate those aspects which are negotiable, getting good agreements not only for providers but also for the clients (deeper details can be found in [10]).

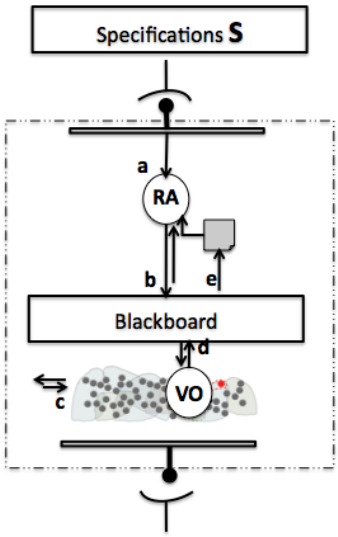


Figure 3. Overview of MACOCO+ receiving a request containing specifications $S$ that must be satisfied by a virtual organization from the service market

Figure 3 shows MACOCO+ receiving a specification $S$ (e.g. *a service capable of sending email with at least a response time less than 200 ms*). The objective of MACOCO is to find an architectural configuration $C$ which satisfy the specifications $S$. A request agent $RA$ wraps the request which contains the specifications $S$ (step a). $RA$ publishes the request into the specific topics (functional categories) of the blackboard (step b), where all the subscribed service providers $SA$ are aware of it. These service providers compete among them to gain the contract, bidding the request by themselves or by creating a coalition with others to bid as a virtual organization $VO$ (step c and d). Each request has its own timeout. When the agent $RA$ reaches the timeout, it closes the offers submission process and it proceeds to evaluate how well the several offers satisfy the specifications $S$ of the request. Internally, a contract

is created between the $RA$ and the services providing the selected configuration $C$. This contract establishes the abstract and concrete specifications ($S^*$ and $S$) and which service agents are implementing the request.

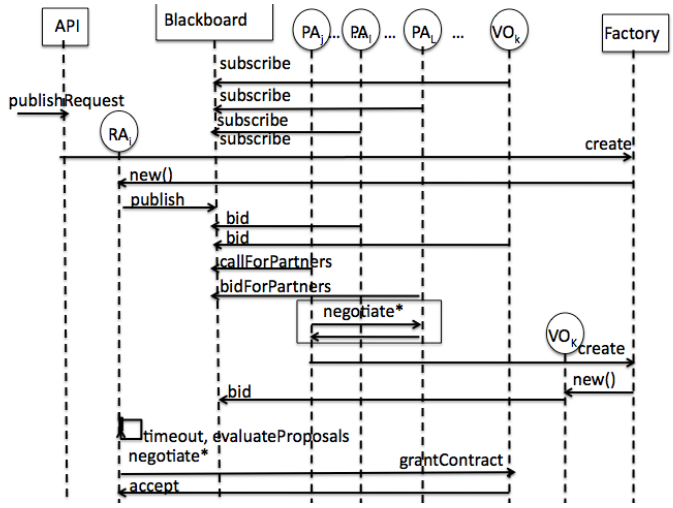


Figure 4. MACOCO+: granting the contract

Figure 4 shows the granting contract process from the beginning to the end. The client *publish a request*, and this request is wrapped into a software agent $RA$. Each request has a different $RA$ on charge. The $RA$ publishes the specifications $S$ into the $Blackboard$ where all the services in the market are already wrapped into an agent and subscribed to it. The request is open to bidding for a specific time given by the client. Because the $Blackboard$ follows a publisher-subscriber design pattern, when an $RA$ publishes a request, all the $SA$ subscribed to the pertinent topics are notified. It is important to mention existing $VO$, serving different requests, are also represented by agents that also have the same goal of $SA$s: *to gain the contract*. More details can be found in [10]. When agents ($SA$s or $VO$s) receive request notifications they can decide to do nothing, to call for partners to bid as part of a virtual organization or to bid alone (see Figure 4). When $RA$ reaches the timeout, it evaluates and ranks the proposals. It negotiates with the top-k agents representing the proposals those aspects which are marked as negotiable. The $RA$ selects a proposal and grant the contract. Because the conditions of the $SA$ or $VO$ agent could change, $RA$ expects the acceptance from the counterpart. MACOCO up to now had not provided adaptation support to clients.

## IV. Proposal

In this section, we present Adaptive, our framework which implements our proposal presented in [5] supporting Service-based systems to drive their adaptation based on market-awareness. Our proposal is quite similar to one of the exemplars presented by Zambonelli [11] in which the awareness was reached by using a blackboard to share the relevant knowledge.

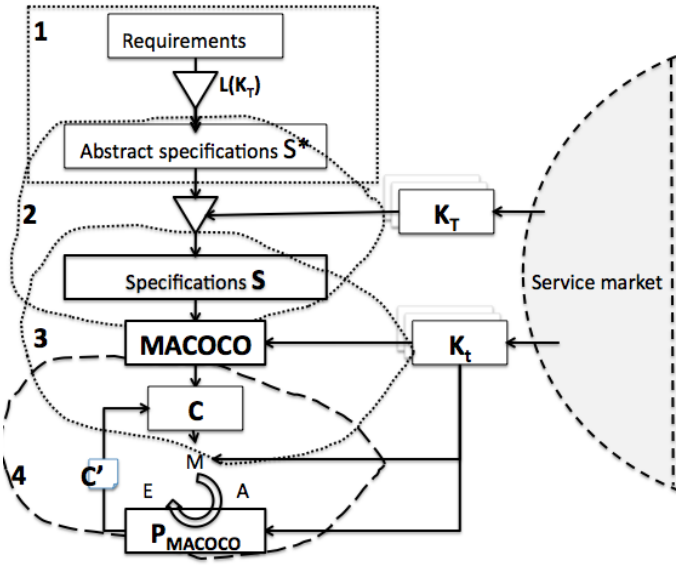


Figure 5. Overview of Adaptive

We use MACOCO+ and our approach to mitigate the obsolescence of the specifications model $S$ (see Figure 5). At runtime, the agreement between clients and providers is stored as a contract which is monitored each time there are new snapshots available and it is recomputed each time a new reasoning of the market is available $K_T$.

The main differences between the Figures 5 and 2 are explained below.

- In the subprocess 3 we have replaced the generic transformation from specifications $S$ to architectural configuration $C$ by MACOCO+, which receives the current snapshot of the market (in order to agents reflect the latest QoS information of the services they are representing) and returns a web service composition (or architectural configuration) $C$, capable to implement $R$ that emerges from the service market.
- In the subprocess 4, we have replaced the component P (Planning component - deeper details can be found in [5]) of the MAPE (Monitoring-Analyzing-Planning-Executing components) feedback loop. Then, in both schemas (Figures 5 and 2), the configuration $C$ is subscribed to a monitoring service in order to track the violations in which could incur $C$.

In both cases the violation history is analyzed by the Analyzer component. If there is enough evidence that $C$ is recurrently violating the specifications $S$, then a reconfiguration is triggered. But, in this case, the reconfiguration calculation $C'$ is obtained by MACOCO+ over the latest snapshot of the market $K_t$.

Each time, a new understanding of the market is produced,

$K_T$, all abstract specifications $S^*$ (including those which are subscribed to the monitoring process) must be remapped into concrete specifications $S$ using the current relevant knowledge domain $K_T$. Then, immediately, the monitoring process checks all the contracts to determine if the current $C$ under $K_t$ is satisfying the new $S$. When a new domain knowledge is available, $K_T$, the process 3 and 4 are automatically re-executed.

Our model also supports requirements evolution. If the requirements are updated, new goals are added, others are removed, priorities change, or quality requirements change, $R$ and consequently $S^*$ are updated by manually re-execute the subprocess 1. Then, all the following subprocesses will be automatically executed.

## V. Implementation

In this section we explain our implementation of the the service market metaphor as a multi-agent system. Moreover, in this section we explain the implementation of *Adaptive*. *Adaptive* is an adaptation recommender system to provide adaptation recommendations to SBSs. *Adaptive* implements the MAPE-K feedback loop. We claim *Adaptive* as a recommender system because up to now, is not implementing the Executing component of the MAPE-K, but instead, it recommends to clients when and which configuration $C$ the client system must apply in order to adapt itself. All the modules of *Adaptive* are implemented in Java and the integration functionalities are exposed as REST web services. The integration between the several modules is through services. In particular, the planner module is implemented under the MACOCO+ model, which is completely implemented in Java over JADE [1] (further details can be found in [9]). In order to maintain the service providers and consumers aware of the market, we shown in Figure 6 how $K_T$ and $K_t$ are obtained. It is important to notice, the market is the module $K$ of our MAPE-K implementation.

Figure 6 shows that the *functional crawler* component collects from different Web-based catalogs the Web service descriptors. The *QoS certifier* component runs a benchmark tool over the endpoint list obtained by the *functional crawler*, in order to gather the QoS measurements (certification snapshots). The *functional clustering* component clusters the Web services (based on their WSDL descriptor files) according to their functionality (if there is not valid categories information available). The *QoS-fuzzy clustering* is the component that clusters each quality aspect of each functionally-equivalent service set into $c$ classes using a modified fuzzy c-means algorithm (deeper details in [12]). In this work we set up $c = 5$.

The Monitoring and Analyzer modules have being implemented up to now using very simple algorithms. The monitoring works with the threshold specified by the clients

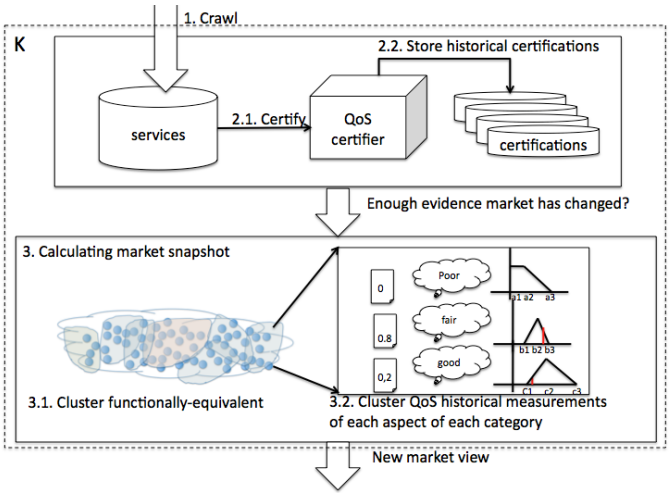


Figure 6. Architecture to produce $K$

(e.g. a client could determine its contract is being violated by providers if the current satisfaction of the requirements has dropped in $\alpha$ percent respect to the previous monitoring step). The analyzer module decides if an adaptation is needed if the number of violations over the total of monitored times of the specific contract is bigger than a $\beta$ percent (also defined by each request).

Figure 7 shows an initial web prototype implementation [2] of *Adaptive* using the as Planner the MACOCO+. This prototype is to support the process 1 of the Figure 5, allowing SBS architects to specify the abstract specification model, which will drive the implementation at runtime. Notice that, each time, the architect selects one quality attribute and constrain using a linguistic variable, *Adaptive* is informing which numerical meaning has that linguistic variable for that quality attribute for that functionally equivalent set in the current service market $K_T$. Then, the request is send to the MACOCO+ module to obtain an initial recommendation to implement $S$.

## VI. Concluding Remarks

We proposed a framework to make both, consumers and service providers dynamically aware of the changes in the market. Consumers are able to specify market-aware specifications that will be eventually concretized in accordance to the real characteristics of the market at runtime. Service providers are represented by active software agents. Agent make service providers become collectively aware of themselves and what clients in the market require. Furthermore, following the behavior of the market and instead of just waiting to be discovered, agents are able to proactively create and maintain virtual organizations that answer to the demands of the market. A virtual organization join forces of different service providers to make better offers (in price

[1] http://jade.tilab.com/

[2] http://cc.toeska.cl/adaptive

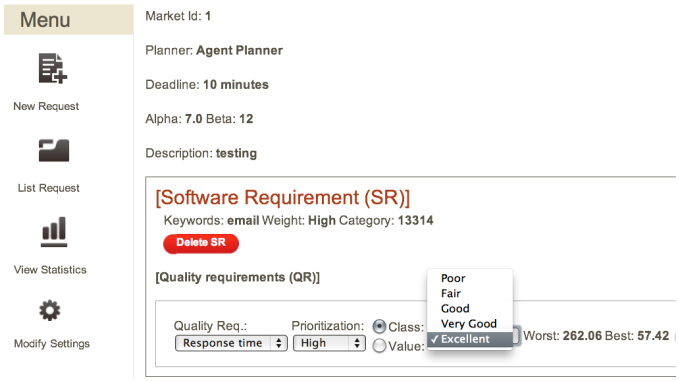


Figure 7. *Adaptive* Web prototype

or even QoS) to clients improving the offers of independent service providers. In this work, we showed how the self-adaptation of SBSs can be reached by implementing services consumers and providers as a service market metaphor.

As ongoing work, we are studying the performance of the approach on a market with thousands of services and hundreds of clients when it is used to allow services organize themselves to serve new requests or maintain current ones.

## ACKNOWLEDGMENT

This work was partially funded by FONDEF (grant D09i1171), UTFSM DGIP 241167 and BASAL FB0821(FB.02PG.11), the EU Marie Curie Project Requirements@runtime and the EU Connect project.

## REFERENCES

[1] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, "Requirements-aware systems: A research agenda for RE for self-adaptive systems," in *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference*, ser. RE '10. IEEE Computer Society, 2010, pp. 95–103.

[2] A. Ramirez, A. C. Jensen, and C. B. H. C., "To appear seventh workshop on software engineering for adaptive and self-managing systems (SEAMS 2012)," in *ICSE*, 2012.

[3] L. Baresi, E. Di Nitto, and C. Ghezzi, "Toward open-world software: Issue and challenges," *Computer*, vol. 39, no. 10, pp. 36–43, Oct. 2006. [Online]. Available: http://dx.doi.org/10.1109/MC.2006.362

[4] P. Zave and M. Jackson, "Four dark corners of requirements engineering," *ACM Trans. Softw. Eng. Methodol.*, vol. 6, no. 1, pp. 1–30, Jan. 1997. [Online]. Available: http://doi.acm.org/10.1145/237432.237434

[5] R. Torres, N. Bencomo, and H. Astudillo, "Mitigating the obsolescence of quality-specification models in service-based systems," in *Model-Driven Requirements Engineering Workshop (MoDRE) (submitted)*, sept. 2012.

[6] R. Calinescu, L. Grunske, M. Z. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic QoS management and optimization in service-based systems," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 387–409, 2011.

[7] R. Torres, H. Astudillo, and E. Canessa, "MACOCO: A discoverable component composition framework using a multiagent system," in *Proceedings of the 2010 XXIX International Conference of the Chilean Computer Science Society*, ser. SCCC '10. IEEE Computer Society, 2010, pp. 152–160.

[8] D. Nguyen, S. Thompson, J. Patel, L. Teacy, N. Jennings, M. Luck, V. Dang, S. Chalmers, N. Oren, T. Norman, A. Preece, P. Gray, G. Shercliff, P. Stockreisser, J. Shao, W. Gray, and N. Fiddian, "Delivering services by building and running virtual organisations," *BT Technology Journal*, vol. 24, pp. 141–152, 2006.

[9] R. Torres, D. Rivera, and H. Astudillo, "From virtual organizations to self-organizing web service compositions," in *Proceedings of the XXIX International Conference of the Chilean Computer Science Society*, ser. SCCC '11. IEEE Computer Society, 2011.

[10] ——, "Web service compositions which emerge from virtual organizations with fair agreements (best student paper)," in *Agent and Multi-Agent Systems. Technologies and Applications*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7327, pp. 34–43.

[11] F. Zambonelli, N. Bicocchi, G. Cabri, L. Leonardi, and M. Puviani, "On self-adaptation, self-expression, and self-awareness in autonomic service component ensembles," in *Proceedings of the 2011 Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops*, ser. SASOW '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 108–113.

[12] R. Torres, H. Astudillo, and R. Salas, "Self-adaptive fuzzy QoS-driven web service discovery," in *Proceedings of the IEEE International Conference on Services Computing*, ser. SCC '11. IEEE Computer Society, 2011, pp. 64–71. [Online]. Available: http://dx.doi.org/10.1109/SCC.2011.87