Model-based dynamic QoS-driven service composition

Antinisca Di Marco, Antonino Sabetta

Model-based dynamic QoS-driven service composition*

Antinisca Di Marco
Computer Science Dept.
University of L'Aquila - Italy
antinisca.dimarco@univaq.it

Antonino Sabetta
CNR-ISTI, Pisa
Italy.
antonino.sabetta@isti.cnr.it

ABSTRACT

As a consequence of their ever increasing pervasiveness in today's systems, software services are expected to guarantee their QoS even when operating in contexts whose operational conditions may continuously change. To cope with such continuous change, services must evolve in a way that is transparent to the end-user. This can be done by exploiting sophisticated means to reason about Quality of Service (QoS) and to drive service construction in a dynamic and automated fashion. The paper tackles this challenging scenario by proposing a model-based framework, called SMART, that automatically constructs complex services with guaranteed QoS. SMART exploits rich service descriptions (in particular concerning QoS characterizations) to automate the negotiation of Service Level Agreements (SLA) and to realize SLA-driven automated service reconfiguration, when the target QoS cannot be achieved by the current service assembly. Finally, SMART addresses the case in which some services involved in the composition do not support SLA negotiation. In this case, monitoring is used to characterize empirically (as opposed to "contractually") the QoS offered by such non-guaranteed services.

1. INTRODUCTION

Software services, provided by different organizations, advertise and offer their functionalities while hiding heterogeneity in the underlying hardware and software infrastructure.

As software services become more and more pervasive, they are required to provide their functionalities guaranteeing a certain level of quality, even when operating in environments whose conditions may continuously change, as in the case of ubiquitous and mobile computing systems.

Due to the high dynamicity and heterogeneity of software-oriented systems and of their execution environment, software services must be capable of evolving in order to guarantee the QoS level stated in a Service Level Agreements (SLA). Achieving this effectively is a challenging problem as the service-oriented software is an "open" world, fully decentralized, and typically no single organization is in control of all the services involved in a system.

The problem of dynamic adaptation to maintain the level of QoS at a desired level is the target of the research presented in this paper. We present a model-driven framework, called SMART, whose aim is to realize dynamic QoS-driven service composition, by building complex, resilient services with guaranteed QoS, by composing simpler, pre-existing services. The service composition occurs in an automatic way by exploiting the SLA negotiation mechanism and SLA-driven service reconfiguration.

The SMART framework aims to improve over the state of the art in three directions. Firstly, it provides the required service by composing services discovered in the network. Whenever a perfect match between the demand and the supply does not exist with respect to QoS requirements, SMART aims at providing one or more service alternatives for the request service by relaxing one or more QoS constraints that the service consumer imposed initially.

Secondly, the SMART framework is meant to work both when the QoS of all the services involved in the aggregation process is guaranteed with a SLA, and in the case that at least one simple service is not guaranteed by SLA. In the latter case, monitoring is used to characterize empirically the QoS offered by non-guaranteed service. From such characterization, SMART synthesizes the Service Level Specification (SLS) that will make up for the lack of the SLA and will be used as input for the QoS and trade-off analysis supporting the SLA negotiation process of the composite service.

Finally, SMART supports monitoring of both individual simple services and the composed service, in order to verify that the SLA of the simple and composed services are satisfied. The monitoring mechanisms alert the framework when either any simple service under a SLA does not respect the negotiated agreement; or the provided composed service shows a worrying trend in its QoS that let suppose that violation of its SLA, or the provided composed service does not satisfy the SLA any more. In these cases, SMART activates reconfiguration mechanisms that evaluate if and how to reconfigure

*This work is partially supported by the EU-funded CONNECT project (FP7–231167) and by the Italian PRIN d-ASAP project.

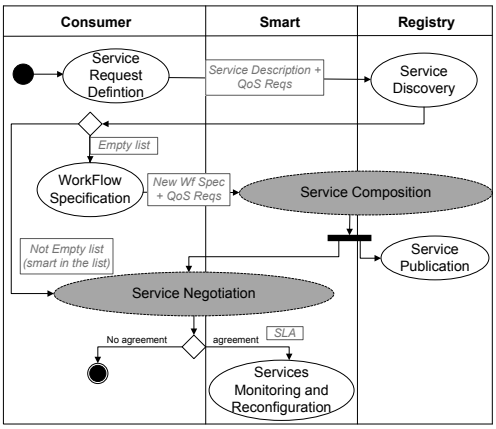

Figure 1: Smart high level operations.

the service, and will implement the selected reconfiguration strategy. The reconfiguration is transparent to the user, as long as the reconfiguration produces a new service having QoS characteristics compatible with the initial SLA.

The remainder paper is structured as follows: Section 2 describes the main features and goals of SMART, whose realization in a reference architecture is described in Section 3. Section 4 gives pointers to some related work and positions our contribution with respect to the existing literature. Finally, Section 5 concludes the paper.

2. SMART FEATURES AND GOALS

The key goal of SMART is to provide a framework for building complex services with guaranteed QoS, starting from simple, pre-existing services. Service composition in SMART occurs automatically, exploiting automatic SLA negotiation capabilities.

In response to user requests, SMART selects, out of a pool of candidates, a set of services that are composed to provide the requested functionalities with the desired QoS level.

To this end, two alternative hypotheses can be formulated:
- All the services that are involved in the overall aggregation process provide a suitable support to the automatic SLA negotiation; in other words the service providers supply the information (functional and extra functional specification of the service) to facilitate the automatic negotiation of QoS.
- At least one of the service that are candidates for the composition does not export a QoS specification and provides no support to automatic negotiation of QoS level.

The SMART framework aims at operating under each of these hypotheses. In order to achieve the latter one, SMART should observe the actual behavior of the candidate services and derive a QoS characterization for each of them *empirically* (as opposed to *contractually*, as in the first hypothesis). Starting from the results of this characterization, a SLS (Service Level Specification) is synthesized. Such SLS, together with the established SLA, are used as input to the analysis supporting the service composition and all together form the assumptions the monitor must check when the composed service is activated.

Figure 1 shows the invocation of SMART described by means of a UML Activity Diagram that specifies, throughout swimlanes, the responsibilities of the activities to be executed. The system responsible of the activities in a swimlane is indicated on top of it. We represent the data exchanged by the activities by means of grey boxes, and complex activities involving many actors throughout grey dashed-line ellipses . A *Consumer* asks for a discovery service to the *Registry*. The request specifies the service in terms of functional and QoS requirements. The Registry returns a list of matching services already published by some Providers. At this point in time, two alternative situations may occur:

The returned list is empty: no existing services are discovered. The Consumer formulates a detailed service request to the SMART framework. The request is composed by the workflow specification, describing the service from a functional point of view, and the QoS requirements the service must satisfy during its execution. SMART dynamically composes the required service by exploiting the on-line QoS analysis techniques to check if it can provide a service with the required QoS. Then, it publishes the new service on the registry and proceeds to the service negotiation. In case the on-line QoS analysis reported a positive response, the negotiation phase is reduced in the SLA exchange. Whereas, in case of a negative response, SMART proposes the consumer several alternatives obtained by relaxing one or more QoS requirements. During this phase, SMART could execute the QoS analysis several times by exploiting the models built in the composition phase. If the consumer does not accept any proposed alternatives, the request is discarded.

The returned list contains Smart as provider of the required service and Smart is selected from the consumer: in this case the service composition is skipped, and the consumer and the SMART framework directly proceed to the negotiation process.

If the negotiation leads to an agreement, SMART sets the monitoring infrastructure to monitor the negotiated SLA. To this end, it monitors both the composed service provided and the assumptions made by the QoS and trade-off analysis during the negotiation phase[1]. Whenever the monitor observes some problems, SMART could decide to reconfigure the service.

SMART must deal with several issues to reach its goals. Among others, the service description is one of the main problems. SMART must manage service description with the different level of details depending on whether the service belongs to the provider (internal) or not (external). In the former case we have a complete model while, in the latter, the only information that SMART considers coming from the service monitoring (SLS) and from the SLA associated to service (if it is a guaranteed one). The service reference model and the devised notation used in SMART have to allow these different levels of detail, making consistent the reasoning made at the different level of details.

[1]Note that, such assumptions are the synthesized QoS characterization (SLS) or the negotiated SLA of the simple services SMART selected to implement the concrete service for the input workflow

Moreover, SMART signs with the consumer a SLA for the composed service. The SLA must be satisfied during the service execution. To this end, SMART exploits reconfiguration mechanisms supported by QoS and trade-off analysis techniques. QoS and trade-off analysis is executed both off-line and on-line, to support reactive and proactive service reconfiguration, respectively. Proactive reconfiguration is made when SMART works to provide a better implementation of a service already provided. In this case, the analysis is done off-line with no real time constraints. Such analysis techniques use complex models with high accuracy. Reactive reconfiguration, instead, timely reconfigures the composed service if the monitoring discovers that the negotiated SLA is violated. Reactive reconfiguration exploits on-line analysis to find the "best" reconfiguration that overcomes quickly the observed problem. On-line analysis works under real-time constraints and uses more simple models. They produce quick estimations but with reduced accuracy. On-line QoS and trade-off analysis is also used during the negotiation phase when a (new) service request is formulated.

Finally, SMART is conceived to integrate with the infrastructure developed in the CONNECT project. CONNECT pursues *eternal interoperability* by synthesizing on-the-fly the connectors through which networked systems communicate. The synthesis process is based on a formal foundation for connectors, which allows learning, reasoning about and adapting the interaction behavior of networked systems at runtime [6]. SMART relies on CONNECT in two respects. Firstly, it reuses some components from CONNECT, such as the monitoring infrastructure, which provides a generic framework that can be easily adapted for monitoring service compositions, and the semantic discovery enabler. Secondly, the semantic discovery component is hooked to more than just a service repository. By exploiting the capabilities of CONNECT, more candidates are added to the set of services available for composition; these candidates are actually CONNECT*ed systems*[2] whose functionalities are provided while hiding protocol and data mismatches using connectors (i.e., protocol adapters and data converters) that are synthesized and deployed automatically.

3. SMART REFERENCE ARCHITECTURE

Figure 2 depicts the main components of the SMART framework and how they are connected to each other. In particular, it is composed by seven components for which we outline the key elements in the following. SMART interacts with the service Registry to discover simple services to compose, and with the Providers of the simple services it selects for the composition. It monitors the services it uses in the composition and the composed service in order to observe any SLA/SLS violation. Its input is the workflow specification enriched by QoS requirements coming from a consumer asking for a (new) service with guaranteed QoS.

Figure 3 shows how the SMART components interact during the Service Composition Process. Service Composition starts from the arrival of a new workflow specification formulated from a consumer. The Service Composition Engine

[2]This term is used in the CONNECT project to indicate systems that are assembled by using a synthesized connector to have two or more networked systems to interoperate.

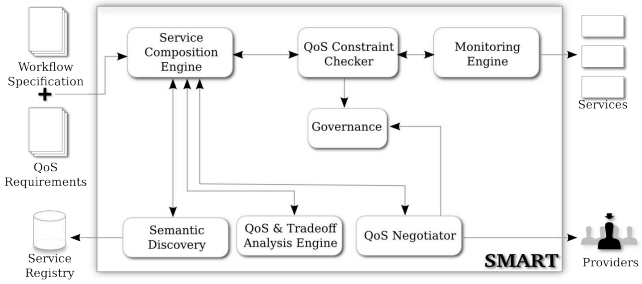

Figure 2: Architecture overview

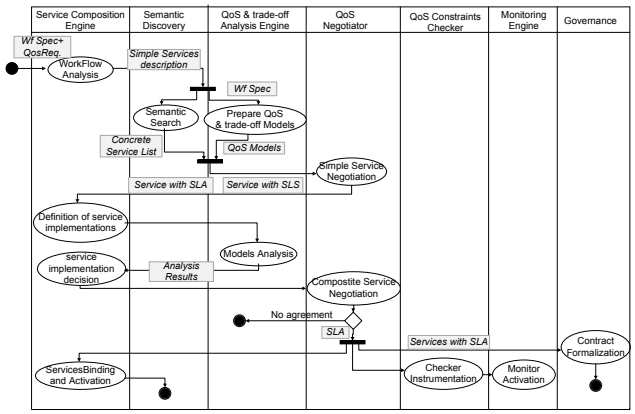

Figure 3: Service Composition Process

analyzes the workflow specification to determine the description of the simple services needed to compose the required assembly. After that, the simple services discovery and the QoS and trade-off models generation for the workflow specification are executed in parallel. The list of the concrete simple service discovered is sent to the QoS Negotatior for the negotiation step. After having contacted the providers of the discovered services, the negotiator returns two lists, the first containing the concrete services for whom the negotiator has been able to finalize a SLA, and the other containing the remaining services. Such information is used by the Service Composition Engine to define possible implementation alternatives of the workflow specification that are passed to the QoS and Trade-off Analysis Engine for evaluation. The analysis results support the composition Engine to decide which alternative to negotiate with the consumer. If the negotiation of the proposed assembly is successful, several actions are executed in parallel: the SLA for the simple services involved in the composition and for the assembly are formalized, the binding to the simple services is executed and the composite service is activated, and the QoS Constraints Checker is set w.r.t. the SLA/SLS of the bound concrete services. Finally, the Monitor Engine instruments and activates the monitors.

Service Composition Engine and Semantic Discovery

The service composition engine takes in input a workflow specification and formulates a service composition that realizes the workflow using services obtained from a repository of available services. Such repository is a front-end to access one or more service registries and to retrieve a set of

service instances (referred to as *concrete services*) that offer the functionalities as required by the workflow.

The registry used in SMART is actually more than just a repository of services, as it wraps and leverages the semantic discovery capabilities offered by the CONNECT infrastructure, and in particular by its *discovery enabler*. Therefore, in certain circumstances (e.g., when a proper "conventional" service is not available), the SMART registry may provide a service that is actually a wrapper for a more complex system, that is obtained by exploiting the ability of CONNECT to synthesise and deploy *mediators*[3] automatically and on-the-fly.

The service composition engine is responsible for the dynamic reconfiguration of the service in order to guarantee that the constraints of the negotiated SLA are respected.

The Service Composition Engine implements two types of reconfiguration: *i)* Reactive, if it is executed in case of violation of the SLA; *ii)* Proactive, if executed as a consequence of the observation of negative trends that could lead to the violation of the SLA.

The service composition engine reasons about acceptable reconfiguration policies available in the service domain, taking into account the constraints and the characteristics of the composition that determine the applicability of the reconfiguration actions (binding changes, migration of services on execution hosts, service duplication, etc..).

The operation of the service composition engine relies on the availability at runtime of a model of the composed service. Such model is used to represent and reason about the properties of the service while the system is in operation. To this end, relevant modeling features are, among others, the workflow specification and its realization via existing services, the offered QoS of these services and of the overall service assembly, and the stipulated contracts (SLA), if available.

QoS and Tradeoff Analysis Engine

Analysis techniques are used in SMART to assist both the Service Composition Engine, in defining a service assembly that matches the target QoS, and the Negotiator, during the SLA negotiation of the composed service.

The QoS and Tradeoff Analysis Engine (in the following, *Analysis Engine*) predicts the QoS that can be achieved by a certain assembly and produces quality trade-offs models. We follow the assume/guarantee paradigm [7] supported by the modeling notations for the development of incremental and compositional analyses, that are suitable for dynamically evolving applications. The validation of this class of applications is performed continuously at runtime through a feedback loop between negotiation/reconfiguration and monitoring mechanisms.

The Analysis Engine's action can be distinguished into:
During the composition phase: the Analysis Engine builds the Qos and trade-off models (one for each QoS attribute of interest and one for the trade-off) of the workflow specification provided by the Service Composition Engine. These models are later parameterized by using the SLA and SLS of the simple services provided by the Negotiator and the Monitoring Engine, respectively.
During the service composition and SLA negotiation phase: the Analysis Engine proceeds to the models parametrization and analysis. Indeed, it produces a set of parameterized analysis models for each implementation alternative the Service Composition Engine defines. The Analysis Engine hence proceeds with their evaluation and, on the basis of the obtained results, the Service composition Engine decides which service implementation to offer to the user. Moreover, if all the considered alternatives cannot guarantee the quality levels as required by the user, this Engine helps formalizing new proposals obtained by relaxing one or more initial QoS constraints.
During the service execution: the same models, fed with monitored parameters, can be used to predict critical situations that can lead to contract defaults. In these cases, SMART is alerted if a suspicious trend or a SLA violation is observed. This mechanism allows the framework to bring the service back to the contract terms.
During the reconfiguration phase: in this case the models, in addition to the modeling of the QoS of the new service that would be obtained by the application of the reconfiguration strategy, include the overhead required by the reconfiguration. The evaluation of such models provides the information on the basis of which the provider decides whether to reconfigure the service or which reconfiguration strategy to adopt.

QoS Negotiator

The QoS Negotiator deals with the process of SLA negotiation. In SMART, the SLA negotiation is executed dynamically when the user requests for the service. It holds the negotiation of both the composed service and the simple services discovered during the composition phase.

Simple service negotiation: the Negotiator receives a list of concrete services, each associated with a service level request and the indication of the corresponding provider. The service level request is formulate as a list of QoS attributes. Each attribute has associated a range of values (in terms of minimum and maximum) the QoS attribute may fluctuate to be considered acceptable.
The aim of the Negotiator is to contact all the providers and negotiate with them the required service level. In general, the Negotiator initially asks for the best service, that is the service having a high quality (by reducing the ranges included in the requests). If the negotiation fails, it repeatedly requires for a lesser performing service since the negotiation is successful.
The output of the negotiation process is composed by two lists containing the concrete service identifications and the SLA the Negotiator contracted for them. Indeed, the first list contains only and all the concrete services for which the Negotiator has reached an agreement, highlighting the ones conform to the request, while the second list contains the remaining ones.

Composed service negotiation: The Negotiator receives the service level request coming from a (new) consumer and

[3] CONNECTors, in CONNECT parlance. A CONNECTor may be used to address different types of mismatches; e.g., protocol mismatches or data-format mismatches.

it starts the negotiation process. To evaluate if SMART is able to afford the new request, the Negotiator asks a prediction to the Analysis Engine sending it the new workload indicated in the request and the QoS attributes of interest. The Analysis Engine, after having parameterized the analysis models with the new workload and with the parameters coming from the monitor, reports its prediction to the Negotiator and the QoS Constraints Checker. If the prediction still satisfies the SLA already negotiated and the new request, the Negotiator proceeds to the SLA formalization otherwise it collaborates with the Analysis Engine and the QoS Constraint Checker to formulate an affordable counterproposal to offer to the consumer. If the counterproposal is accepted by the consumer, they proceed to the SLA formalization, otherwise the request is discarded.

QoS Constraints Checker

The QoS Constraints Checker matches the end-users quality expectations formalized in a SLA and expressed as application-specific QoS metrics, with a monitoring infrastructure that is able to observe properties defined in terms of those metrics. In practice this requires a translation of high-level specifications of QoS objectives, defined in terms of business metrics, into constraints that are expressed in terms of readily observable events and metrics. After having made the matching and from the corresponding output, it instruments the Monitoring Engine to allow the observation of the events and the metrics identified. Finally, it checks that the finalized SLA are not violated by capturing all events generated by the monitoring. In case it observes any violation, it alerts the Service Composition Engine to evaluate the opportunity of a service reconfiguration and the Governance to manage the legal and the administrative aspects as stipulated in the signed SLA.

Monitoring

Monitoring in SMART plays an important role both at the service composition stage and, later on, when the services that participate in the composition are observed to ensure that the QoS goals are met.
The automatic composition of services is based on models describing both the individual services that are candidates for the aggregation and on models describing the result of this aggregation, i.e., the composite service to which the end user is directly interested. As these models are typically parametric, the monitoring is used to complete them by providing measured values for their parameters.
In particular, the framework must be able to cope with scenarios in which the service providers do not provide models of the operational characteristics of offered service or in general do not support the automatic negotiation of SLAs. In this case, a QoS model of the services to be composed models is inferred by direct observation.

Concerning the phase when a service composition is established and operational, and a SLA is reached, deviations from the expected behaviour must be detected as soon as possible. Actually, this is not only necessary to ensure that SLAs are not violated, but also to carry out proactive adaptation by adopting a different configuration in order to *prevent* violations that are considered likely to occur in the near future. The key principle is that dynamic reconfiguration of the aggregated service must happen in a transparent way to the end user.

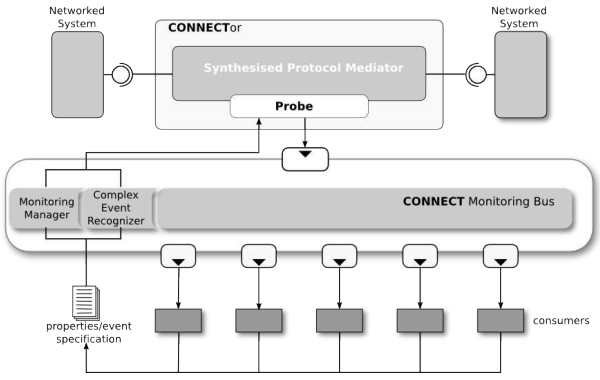

Figure 4: Connect monitoring infrastructure

The monitoring infrastructure used in SMART is borrowed from the CONNECT Monitoring Framework, whose high-level architecture is outlined in the following. The very vision of CONNECT, i.e., achieving automated and eternal interoperability puts on-line approaches, and therefore monitoring, in a central position in the overall project. Monitoring in CONNECT is involved in dependability assurance activities, but it also contributes to bridging the gap between existing approaches to behavioural learning and CONNECTor synthesis – originally conceived for off-line use – and the CONNECT world, where everything happens dynamically and thus requires approaches to work in an on-line fashion.

In CONNECT, monitoring is conceived as a common core service offered to the other Enablers to implement feedback loops whereby approaches to dependability analysis, CONNECTor synthesis, behaviour learning can be applied to an on-line setting and can be enhanced to cope with change and dynamism. Monitoring is performed alongside the functionalities of the CONNECTed System and is used to detect conditions that are deemed relevant by its clients (i.e., the other CONNECT Enablers). Upon detecting one such conditions, the monitoring system alerts the interested client which, in turn, triggers an update of the analysis, synthesis, learning respectively. In this way, powerful but expensive techniques are executed only when necessary.

The monitoring infrastructure in the CONNECT project is realized as a generic, flexible infrastructure that decouples business-level (or high-level) event specification from the underlying observation and detection mechanisms. From a technical viewpoint, this decoupling is achieved by exploiting a message-oriented backbone, conforming to the JMS standard, to which *probes* push raw, low-level event occurrences. These are then picked-up by a complex event recognizer (implemented using the JBoss Drools Fusion rule engine [3]) to perform complex event processing (CEP [9]). High-level events, once detected, are reported back to the interested clients, again using the message-oriented JMS backbone.

Governance

The focus of SMART is on the *technical* support for automated service composition and QoS negotiation; however the realization in practice of such a framework requires to be grounded on a set of mechanisms, rules, policies, and procedures that may relate to non-technical aspects. The need for such mechanisms, which are generally referred to

as "SOA governance", is widely recognized in the service-oriented community [14]. SMART itself relies on a governance infrastructure, e.g., to ensure legal enforcement of contracts, beyond their technical realization, however, a discussion of these aspects is beyond the scope of this paper. The interested reader is referred to [2] for an introduction to the concepts underlying the notion of SOA governance.

4. RELATED WORK

In literature a lot of work can be found related to the single aspects SMART integrates. Among others, interesting approaches to service composition or reconfiguration supported by extra-functional analysis are defined in [4, 15] whereas [1, 10, 13, 16, 5] envision approaches to SLA monitoring, negotiation and representation.

Although many contributions describe approaches for single aspects in SMART, quite a few presents a thorough approach. MUSIC project [8] developed a middleware managing the service adaptation required by their mobility. It monitors the context and the resources to catch their changes and adapts the service to fulfill the users' QoS requirements. The approach uses QoS predictors and utility functions to support the adaptation process. The adaptation is based on the concept of service plan, i.e. a platform-independent specification containing information on service configurations, its dependencies on the environment and its QoS. Different from our framework, MUSIC does not include negotiation process and SLA management. Moreover, the service adaptation is executed since an experienced mobility caused the change in the context and hence in the available resource.

The goal of the CASCOM project [11] is to create a coordination infrastructure that combines semantic service discovery, intelligent context-aware agents, dynamic service composition and execution across both mobile and fixed peer-to-peer overlay networks. Several points in the research agenda of CASCOM are closely related to SMART, although the underlying technical solutions may be very different. Also, the emphasis of CASCOM is especially on artificial-intelligence approaches and P2P networks, whereas a key idea of SMART is model-driven SLA negotiation and proactive reconfiguration to achieve the target QoS.

5. CONCLUSION

As service-oriented systems gain popularity and become more and more pervasive, the need for ensuring that they provide their functionalities within precise quality levels becomes more and more important. The open-world assumption that is entailed by the very service-oriented vision makes unpredictable heterogeneity and continuous change two key challenges that must be addressed in order for service-oriented systems to succeed in delivering the expected functionalities with guaranteed QoS levels.

This paper presented a framework to tackle these challenges. SMART leverages approaches investigated in the context of the CONNECT project, from which, among other elements, we borrow the capability of hiding protocol and data-format mismatches through automated on-the-fly connector synthesis. In order to ensure QoS in the face of changes, SMART combines automated SLA negotiation, proactive reconfiguration of service compositions based on monitoring and model-based reasoning, and dedicate online QoS analysis techniques. We presented the overall framework, and explained the high-level responsibilities of each building block in the architecture.

Although some of the building blocks of the framework are already available (e.g., off-line QoS analysis tools [12]), and some others are being developed (e.g., the CONNECT monitoring infrastructure), the implementation of the overall SMART framework is planned as a future work.

6. REFERENCES

[1] A. Bertolino, G. De Angelis, A. Sabetta, and S. G. Elbaum. Scaling up SLA monitoring in pervasive environments. In *ESSPE*, pages 65–68, 2007.

[2] Todd Biske. *SOA Governance*. Packt Publishing, 2008.

[3] P. Browne. *JBoss Drools Business Rules*. Packt Publishing, 2009.

[4] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola. Qos-driven runtime adaptation of service oriented architectures. In *ESEC/SIGSOFT FSE*, pages 131–140, 2009.

[5] M. B. Chhetri, J. Lin, S. Goh, J. Y. Zhang, R. Kowalczyk, and J. Yan. A coordinated architecture for the agent-based service level agreement negotiation ofweb service composition. In *Proceedings of ASWEC*, pages 90–99, 2006.

[6] CONNECT Consortium. Connect: Description of Work, 2009.

[7] O. Grumberg and D. E. Long. Model checking and modular verification. *ACM Trans. Program. Lang. Syst.*, 16(3):843–871, 1994.

[8] IST-MUSIC Project. Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments. http://www.ist-music.eu/.

[9] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[10] H. Ludwig, A. Dan, and R. Kearney. Cremona: An architecture and library for creation and monitoring of WS-agreents. In *Proceedings of ICSOC 2004, New York,USA, November 2004*, pages 65–74. ACM.

[11] M. Schumacher, H. Helin, and H. Schuldt. *CASCOM: Intelligent Service Coordination in the Semantic Web*. Birkhäuser Basel, 2008.

[12] SEALAB-QOS Group of University of L'Aquila. MOdel driven conStruction of QUeuIng neTwOrks. http://sealabtools.di.univaq.it/MosquitoHome.html.

[13] J. Skene, D.D. Lamanna, and W. Emmerich. Precise Service Level Agreements. In *Proc. of ICSE 2004*, pages 179–188. IEEE Computer Society Press, 2004.

[14] P. J. Windley. SOA governance: Rules of the game. on line at http://www.infoworld.com, Jannuary 2006.

[15] T. Yu, Y. Zhang, and K.-J. Lin. Efficient algorithms for web services selection with end-to-end QoS constraints. *ACM Trans. Web*, 1(1):6, 2007.

[16] F. Zulkernine, P. Martin, C. Craddock, and K. Wilson. A policy-based middleware for web services sla negotiation. In *Proceedings of ICWS '09*, pages 1043–1050. IEEE Computer Society, 2009.