Service Composition (re)Binding Driven by Application–Specific QoS

Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, Francesco Perfetto, and Maria Luisa Villani

RCOST - Research Centre on Software Technology
University of Sannio
Palazzo ex Poste, Via Traiano 82100 Benevento, Italy
{canfora, dipenta, r.esposito, villani}@unisannio.it, ggperf@libero.it

Abstract. QoS–aware service composition and binding are among the most challenging and promising issues for service–oriented architectures. The aim of QoS–aware service composition is to determine the set of services that, once composed, will perform the required functionality, and will best contribute to achieve the level of QoS promised in Service Level Agreements (SLAs). While the existing works focus on cross–domain QoS attributes, it would be useful to support service composition and binding according to some characteristics on the borderline between functional and non–functional attributes, often specific to the service domain.

The paper describes a QoS evaluator that, integrated with our previously developed binder, allows the use of application specific QoS attributes for composite service binding and re–binding. The applicability of the proposed approach and tool is shown through a case study related to the image processing domain.

Keywords: Quality of Service, Dynamic binding, Re–binding, Composite Web Services.

1 Introduction

Late, dynamic binding of service compositions constitutes one of the most interesting and relevant challenges for service–oriented architectures. In this scenario, a service composition can contain some abstract service specifications – e.g., indicating that a hotel booking service is needed at a particular point of the workflow – without specifying the binding to some existing services. When the functionality offered by more available services is equivalent, the binding is driven by some non–functional, Quality of Service (QoS) criteria, such as minimizing the cost, the time or achieving a tradeoff between the two.

In case the bindings need to fulfill some global constraints imposed over the workflow and (near) optimize a global fitness function, proper aggregation formulae have been proposed to estimate the QoS of the composition from the QoS attributes of invoked services and from some properties of the workflow [4]. Finding a solution of the aforementioned problem is NP–hard: this was addressed by

A. Dan and W. Lamersdorf (Eds.): ICSOC 2006, LNCS 4294, pp. 141–152, 2006.

various authors: Cardoso *et al.* [5] and Zeng *et al.* [11] proposed to use Integer Programming, while, in past work, we used Genetic Algorithms (GAs), that resulted to be more scalable and allowed the use of non–linear QoS aggregation formulae [2].

While most of the existing works focus on cross–domain QoS attributes – e.g., response time, cost, availability, reputation, reliability, etc. – it would be useful to also consider other attributes, that are specific of the service domain/purpose. For example, if the service composition returns a photo, one would try to maximize the photo resolution or the number of colors, keeping low, if possible, the cost and the response time. Much in the same way, a travel booking composite service – involving flight and hotel booking – would try to achieve a compromise between the cost and the hotel category, favoring hotels and airlines encompassing the maximum possible number of priority clubs.

This paper presents the use of application–specific QoS attributes for the purpose of QoS–aware composition and re–binding. In particular, this work enhances our framework for service (re)binding [3] by:

- introducing constructs, and a tool, to define new QoS attributes (specifying type, scale and domain) and to annotate services with these attributes;
- specifying a language for defining QoS attribute aggregation formulae, i.e., formulae similar to those defined by Cardoso [4] for domain–independent attributes. The formulae specification is supported by a guided editor and a type–checker;
- realizing an interpreter for the aforementioned formulae; the interpreter is integrated in our service binder described in [3] and is used to evaluate the QoS of a composite service at binding–time and at run–time (to trigger re–binding if necessary).

The remainder of this paper is organized as follows. Section 2 defines the language for specifying QoS attributes and aggregation formulae. Section 3 describes the QoS–attribute definition tool, highlighting its architecture, its features and how it is integrated with our binder. Section 4 shows the approach at work in the context of a image processing workflow. After a review of the literature in Section 5, Section 6 concludes the paper and outlines the directions for future work.

2 QoS Definition Language

Let us consider a composite service S of n *abstract services*, $S \equiv \{s_1, s_2, \ldots, s_n\}$, whose structure is defined through a workflow description language (e.g., WS–BPEL). Each abstract service s_i can be bound to one of the m *concrete services* $cs_{i,1}, \ldots, cs_{i,m}$, which are functionally equivalent, while exhibiting different QoS values. As said in the introduction, the choice of bindings can depend on an objective function and on a set of constraints. Determining the (near) optimal solution requires to evaluate each solution, estimating the QoS of a concrete workflow, i.e., bound to a set of concrete services. Cardoso *et al.* [5] defined QoS aggregation formulae for each pair QoS attribute–workflow construct. For example, the cost (or

the response time) of a sequence of service invocations is given by the sum of each cost (response time), while the cost of a *switch* is given by the weighted sum of costs for each *case*, where weights are the probabilities that cases will be followed.

In most cases, the aforementioned aggregation formulae are cabled in the optimization algorithm the binder is using. However, as mentioned in the introduction, in many cases it is useful to consider QoS attributes, sometimes specific of a particular domain, sometimes specific of a particular application, for which the aggregation formulae have not been defined yet. Therefore, it is necessary to provide a language and a tool to specify aggregation formulae, and to allow the QoS–aware binder to interpret such formulae for estimating the QoS of the whole composition. To this aim, we developed a language that permits to specify a new QoS attribute, defining:

1. *The type:* supported types are primitive types (integer, real, Boolean), strings and collection types (*Set*, *Bag* and *Sequence*). For integer and real it is either possible to define a range of possible values, or to specify an enumeration of admissible values. For strings it is necessary to enumerate values (thus imposing an order relationship among them). Collection types can be used when the QoS value for a service is constituted of sets of atomic values. In particular, *Set* indicates the mathematical set (no order relationship, no repeated values), *Bag* admits repetitions and *Sequence* imposes an order relationship. The chosen type limits the set of operations that can be used when defining the aggregation formulae. For example, a set supports operations such as union, intersection, while it is not possible to apply arithmetic operators. If necessary, it is possible to get the set cardinality and then apply on it any operator supported for the integer type.
2. *The scale:* ordinal, interval, ratio, absolute. As for the type, the scale limits the set of admissible operations. Since the QoS attribute must be able, at least, to establish an order relationship between two services (i.e., indicate which service is better from a particular QoS perspective), the nominal scale is not considered.

For example, if we consider the photo domain, the *color depth* (defined as the number of bits encoding colors) QoS attribute is of type integer and its scale is ordinal. For any service involving a payment, the *accepted credit cards* attribute is a set, containing strings indicating the various credit cards accepted. The scale for this type of attribute is the ordinal scale where the order relationship is defined over the set cardinality (i.e., the more credit card are accepted, the better is the service). Finally, the *refresh rate* attribute of a webcam service can be considered a real value in the ratio scale.

Our approach for specifying types is similar to what available in the WSLA language [6]. However, WSLA does not consider Collection types nor it defines how QoS attributes values can be aggregated. Similarly to Cardoso, who defined aggregation formulae for domain–independent attributes (cost, response time, etc.), we can compute overall workflow QoS, specifying, for each workflow control construct, aggregation formulae for domain-specific attributes too. In order to

Table 1. Aggregation formulae for some domain–specific QoS attributes

Attribute	Workflow construct	QoS aggregation formula
Color Depth	Sequence	$min(Ai)$
	Switch	$maxProbability(Ai, pi)$
	Flow	$min(Ai)$
	Loop	Ai
Credit Cards	Sequence	$intersection(Ai)$
	Switch	$maxProbability(Ai, pi)$
	Flow	$intersection(Ai)$
	Loop	Ai
Refresh Rate	Sequence	$min(Ai)$
	Switch	$sum(pi \cdot Ai)$
	Flow	$min(Ai)$
	Loop	Ai

obtain that, the language we propose offers a set of operators and functions, most of them inherited from the Object Constraint Language (OCL) [9]. In particular, the language includes mathematical operators, Boolean operators, collection operators, and finally *keywords* proper of the aggregation language. These indicate parameters to be used in aggregation formulae, i.e.:

- k, the number of iterations for a *Loop*;
- pi, the probability of following the i–th case in a *Switch*;
- Ai, the QoS of the inner node of a workflow constructs. For a *Sequence*, Ai is the array of QoS for nodes belonging to the sequence; for a *Switch* it is the array of QoS for all cases; for a *Flow* it is the array of QoS for all the children; for a *Loop* is the QoS of the *Loop* inner node.

Table 1 shows examples of aggregation formulae for some domain–specific QoS attributes, i.e. *color depth* of a photo service, number of *credit cards* accepted from a payment service and *refresh rate* of a temperature service. For the *Sequence* and the *Flow*, the *color depth* is the minimum among the values Ai of the inner nodes. For the *Switch* it can be defined as the color depth Ai for the case having maximum probability pi. Finally, for the *Loop* it is just the value computed for the inner node. For the *credit card* attribute, the aggregation is made through the set intersection operator over *Sequences* and *Flows* (i.e., considering the set of credit cards common to all the services), while for a *Switch* it is the value of the branch having highest probability, and for the *Loop* the value computed for the inner node. The *refresh rate* aggregates similarly to *color depth* for all workflow construct, except for the *Switch*, where an averaged weight is computed. Finally, it is worth to point out that, even in our example we considered only the most common workflow constructs, the language can be used to define formulae for further constructs.

Besides the aggregation formulae, it is necessary to specify the function to be used for evaluating the attribute and to impose an order relationship between services (thus permitting their comparison). Let A be the value of our QoS attribute for a given service. For attributes (e.g., *color depth*) for which the type already imposes an order relationship, the function is the identify function, while

for *credit card* – for which the order relationship is not imposed – the function is $size(A)$, where the function *size* returns the cardinality of the set A.

As detailed in Section 3, the formulae specification is supported by a guided editor and a type–checker.

3 The QoS Aggregation Tool

The workflow QoS-aggregation mechanism has been implemented as part of the WS–Binder Tool [3]. The previous release of the binder supported composite service (re)binding considering cross–domain attributes such as Cost, Response Time and Availability. As shown in Fig. 2, the binder has been extended with the following modules:

- *A QoS Aggregation Function Editor:* it is a web–based editor (see Fig. 1) that a service integrator can use to define new QoS attributes and their aggregation formulae.
- *A Type Checker:* at design time, the *Type Checker* is used to verify the type–correctness of the aggregation formulae specified by the service integrator, according to the language rules and the type scales.
- *A QoS Formulae Interpreter:* at run–time, given a composite service workflow and a possible set of bindings, the *Interpreter* evaluates the workflow global QoS.

The whole environment is realized in Java. Services are deployed using the Axis container[1] while WS–BPEL composite services are executed using the *ActiveBPEL* engine. The QoS aggregator is also realized in Java using the Java Compiler Compiler (JavaCC) Parser Generator[2]. The tool GUI has been developed using the JSP technology.

3.1 Development Time

When designing a composite service abstract workflow, the system integrator may want to specify an objective function and some QoS constraints that will drive the binding. It can happen that either the objective function or the constraints involve some application–dependent QoS attributes. To support binding based on these attributes, it is necessary to have aggregation formulae defined for them. From the preference settings user interface, the system integrator may select the attributes of interest for the composition. The tool provides a support to (i) choose and insert the attributes to be considered for the objective function and (ii) define constraints for some of the attributes, according to their type.

In addition, the user may decide to add new attributes. Of course, this is needed whenever, in the context of a composition, a QoS attribute has to be considered, for which an aggregation function has not been defined before. In

[1] *http://xml.apache.org/axis*
[2] *https://javacc.dev.java.net/*

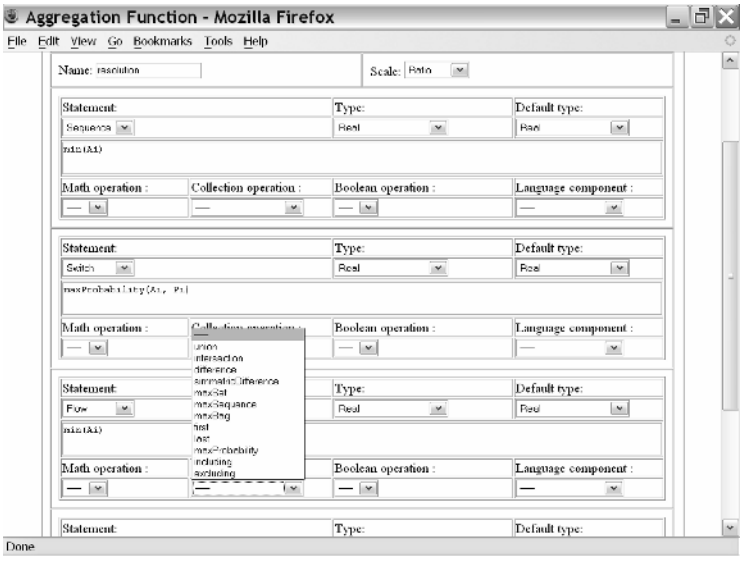


Fig. 1. QoS Aggregation Function Definition Interface

this case, its definition could be specific for the particular composite service being designed. A service integrator can, for example, define his/her own way to aggregate image resolution, while others could do it differently for services having different purposes. Finally, it can happen that the attribute definition is more generic, thus reusable for other service compositions within the same (or related) domain. This is especially the case when attributes are defined by domain experts.

To add new QoS attributes, and to specify aggregation formulae using the language described in Section 2, a guided editor is available. Fig. 1 shows a screenshot of the editor interface. Given the QoS attribute, with the indication of scale and type, the user is required to edit a function for each workflow construct, in a guided fashion. This is achieved by specifying the return type and the aggregation formula.

3.2 Binding Time

At binding time, the QoS Formulae Interpreter allows to estimate the QoS of a concrete workflow (i.e., a workflow for which the abstract services have been bound to some possible concrete services). This is done by applying the defined aggregation formulae over the workflow topology and the QoS values of the services composing it. This permits, using optimization techniques such as those defined in [2], the QoS-aware (re)binding of a workflow according to domain–oriented attributes. In this case, the QoS Aggregator component is used by the Binder in the selection process of the solution services to the optimization problem. The next subsection explains their interaction.

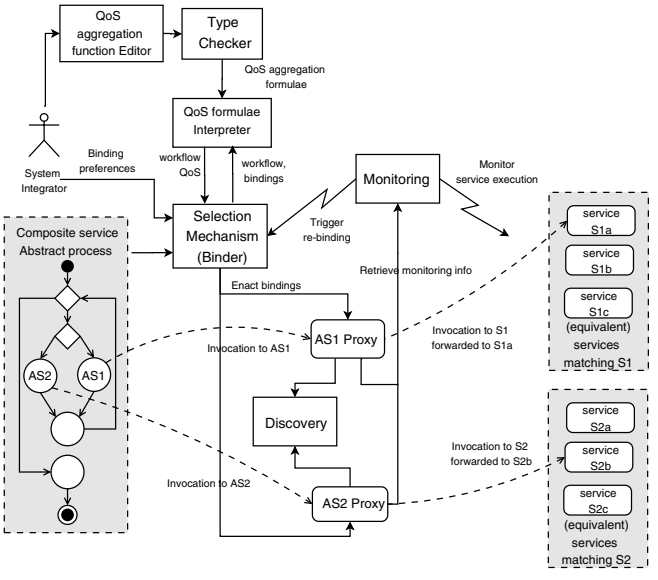


Fig. 2. WS-Binder extended architecture

Integration with the WS–Binder. Fig. 2 shows the extended architecture of the WS–Binder. Specifically, the abstract workflow is a WS–BPEL process definition containing invocations to proxy services. These represent the abstract services and are used to realize the bindings with the final services at run–time and allow re–binding. Indeed, just before the execution, each proxy service allows to retrieve, through some discovery mechanism, and maintain a list of candidate services for the binding, together with their QoS information. This consists of estimated values from monitoring data for attributes like response time and availability, and declared values by the service provider at publication time for the other QoS characteristics of each service. These lists of services are passed to the Binder to determine the (near) optimal concretization for the abstract workflow. In our tool, this is accomplished using GAs, as described in [2]. The genome is represented by an integer array with a number of items equals to the number of distinct abstract services present in the process specification. Each item, in turn, contains an index to the array of the services matching that abstract service. The two–points crossover and a mutation operator that randomly changes a binding are used to generate new individuals. In this generation process towards convergence, the QoS Aggregator module is used to evaluate the individuals. Indeed, the individuals with the best value of the fitness function will reproduce. The fitness function for a genome g is:

$$F(g) = \sum_{i=1}^{n} (w_i \cdot V_i(g)) + w_d \, D(g) \tag{1}$$

where $V_i(g)$ is a normalized value, in the interval $[0, 1)$, of the attribute Q_i for the workflow[3]. Each w_i in (1) is a real, positive weight indicating the importance a service integrator (or user) gives to the attribute Q_i of the fitness function, while $D(g)$ is the distance of the fitness value from the constraint, and w_d weights the penalty factor. Once a solution to the composition optimization problem is found, the bindings are communicated to the proxy services and the process execution may start. When invoked by the engine, the proxy services forward the invocation messages to the services bound and permit to monitor them, e.g., the response time and availability. The re–binding trigger follows the workflow execution to detect and issue re–binding needs. To this aim, the QoS formulae Interpreter component is continuously used to update the QoS estimations at each step of the workflow. A possible re–binding will imply the execution to be suspended, new bindings computed again by the Binder on the workflow slice that remains to be executed, and the old bindings updated through the proxy services. Thus, the execution will continue. The final result for each QoS attribute considered is returned at the end of the process execution.

4 Case Study

This section presents the approach at work over an image manipulation process. The process (shown in Fig. 3) takes as an input one or two images, plus some options. In case a rotation is requested, the image is properly rotated. Then, the *addConstant* operation makes changes to the image basic colors, while the *executeMedian* smoothens the image. Subsequently, a sum, or a difference (e.g., adding a frame or removing a background) is computed with the second image. Finally, the image is properly scaled. The QoS attributes considered for this process are:

1. *the cost*, with aggregation formulae defined as in the paper [3];
2. *the color depth* (values contained in the enumeration $\{16, 24, 32\}$ bits), having aggregation formulae defined in Table 1; and
3. *the resolution*, in terms of image number of pixels, having aggregation formulae similar to color depth.

As a first step, we evaluate how the GA is able to search for a (near) optimal solution according to a given fitness function and a constraint set. Let us suppose one wants to maximize *resolution* and *color depth* while keeping $cost \leq 11$. We set our GA with a population of 50 individuals, 100 generations, a crossover probability of 0.7 and a mutation probability of 0.01. Fig. 4 shows how the fitness (a), the cost (b), the resolution (c) and the color depth (d) evolve over the GA generations. In particular, Fig. 4(a) shows the averaged fitness over 30 runs of the GA, indicating how the fitness is able to drive the search towards a (near) optimal solution.

[3] Note that attributes are normalized (see Zeng *et al.* [11] for details) so that higher values of $V_i(g)$ always correspond to better QoS.

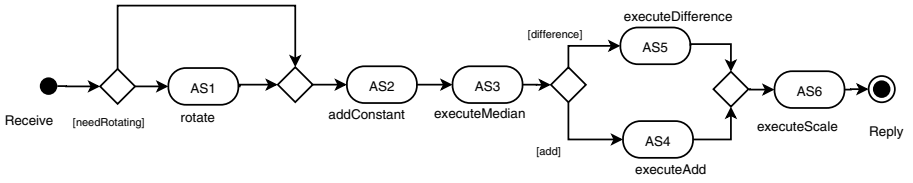
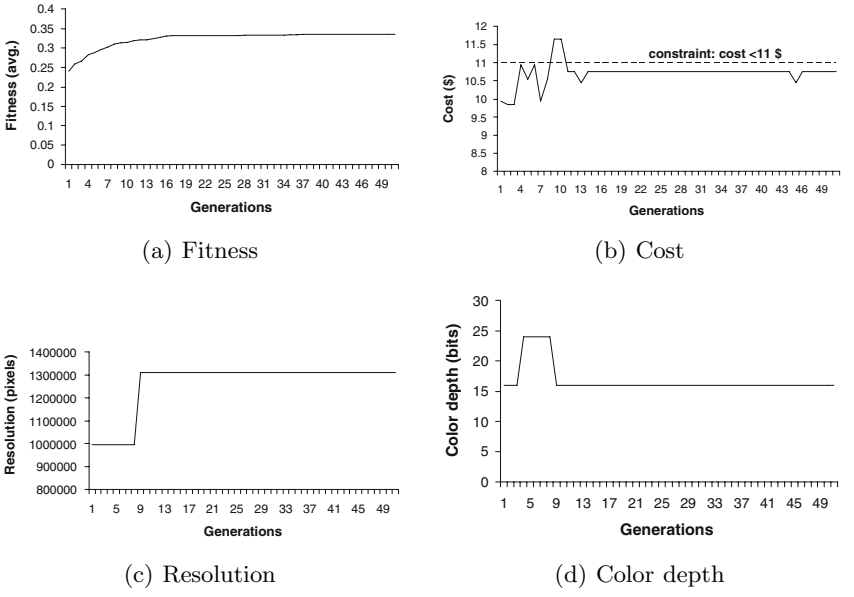


Fig. 3. Image transformation process



(a) Fitness (b) Cost

(c) Resolution (d) Color depth

Fig. 4. GA evolution

The other figures show how, for a particular run of the GA and the best individual, the different QoS attributes evolve. After 5 generations, the algorithm tries to increase the color depth from 16 to 24 bits. This produces an increase of the cost, however still within the constraint. After 9 generations, a solution with a higher resolution is found. However, this produces an unacceptable increase of the cost, violating the constraint. Alternative solutions are therefore selected: the resolution is kept high, while accepting to reduce again the color depth. In this particular case, the weights chosen on the fitness function and the tradeoff between the resolution increase and the color depth decrease balances in favor of a better resolution (reached at generation 10).

As described in [3], it may happen that the actual QoS measured at run–time differs from the initial estimates. For example, when estimating the overall QoS of the image processing workflow shown in Fig. 3, the cost of the *rotate* service does not highly contribute to the overall cost, since the probability of executing it, declared from the service provider, is of 20%. Because of that, the workflow

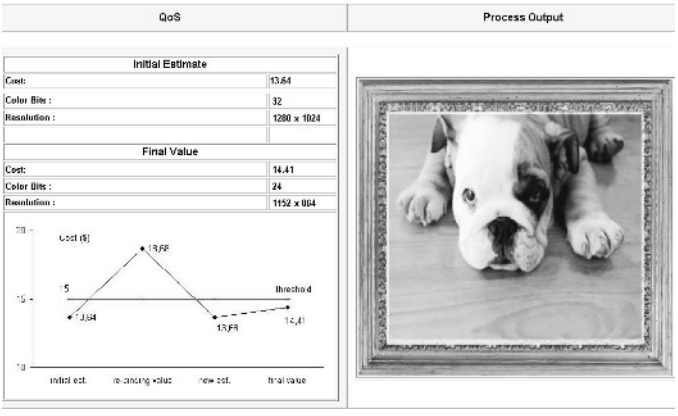


Fig. 5. Process monitoring and output

is bound to a set of services that guarantees an overall cost of 13.64 $, within the constraint of 15 $ imposed in this case, a resolution of 1280×1024 (i.e., 1.3 M pixels) and a color depth of 32 bits. However, it happens that, when the user executes the process, s/he decides to rotate the image. After executing the *rotate* service invocation, the overall cost is re–estimated, indicating that the constraint imposed over the cost is going to be violated. This triggers a re–binding over the slice of the workflow still to be executed (see [3] for details). In particular, two abstract services were re–bound:

1. *rotate (AS1):* from a service having cost=8.40 $, color depth=32 and Resolution= 1280×1024 to a service having cost=4.40 $, color depth=24 and Resolution= 1152×864;
2. *executeMedian (AS3):* from a service having cost=2.80 $, color depth=32 and Resolution= 1280×1024 to a service having cost=1.80 $, color depth=24 and Resolution=1152×864.

The new bindings guarantee a cost within the constraint (13.88 $), while lowering the resolution at 1152×864 and the color depth to 24 bits. Details on the QoS initial estimates, the final QoS values, and the dynamics of the cost attribute (i.e., initial estimate, run–time estimate triggering the re–binding, new estimate and final value measured) are shown in the monitoring view of WS–Binder (Fig. 5), together with the output, i.e., the picture produced by the process.

5 Related Work

To support a QoS-aware composition, models and techniques for workflow QoS estimation and optimization are being developed. In [4,5] a mathematical model is proposed for workflow QoS computation, using metrics aggregation functions which are defined for time, cost, reliability and fidelity. In our work, we propose

to precisely identify domain–wide attributes in order to have consistent ways to aggregate them within workflows.

Aggarwal *et al.* [1] focus on the QoS–driven selection and composition features of the tool METEOR-S. QoS attributes are numerical and formally defined as an ontology that represents generic metrics is used, which also includes the concept of domain-specific QoS metrics. To express process–level QoS constraints, the aggregation operator must be specified for each attribute. The objective function for optimization is a linear combination of the parameters and solved through an integer programming tool, which outputs a set of feasible (sub)optimal solutions, among which the service integrator may choose. In our knowledge, this is the first work where it is explicitly foreseen the possibility of defining domain–specific QoS attributes. Nevertheless, the case studies reported in the work are limited to cross–domain attributes and it is not discussed how different domain–specific QoS attributes can aggregate over workflow constructs. The same authors [8] mentioned that the integrator could specify how the global value for a QoS attribute is computed for a specific process. Differently from them, our approach does not require to necessarily specify aggregation formulae for each process: it would only suffice to define aggregation formulae for pairs QoS–attribute/workflow constructs. Then, the estimated QoS for the whole process is computed automatically. Zeng *et al.* [11] focus more on the optimization problem for workflow bindings based on QoS criteria, which is solved through integer programming techniques. Another work on these issues is by Yu and Lin [10], where a different optimization algorithm is presented. Serhani *et al.* [7] propose a QoS broker-based architecture to support the client in selecting web services based on his/her required QoS.

6 Conclusions

QoS–aware composition and binding represents a challenging mechanism for service–oriented architectures. This paper describes how such a composition can involve not only cross-domain QoS attributes, but also attributes specifically defined for a particular domain or even for a particular application. In that case the service integrator can define domain–specific attributes together with customized aggregation formulae. This permits to estimate the attribute value over a workflow, to determine the (near) optimal bindings and, if necessary, trigger re–binding at run–time.

Work–in–progress is devoted to apply the proposed approach to further case studies and to exploit it to automatically generate test cases, using evolutionary testing techniques, with the aim of violating the SLA in case the latter includes some constraints over domain–specific QoS attributes.

Acknowledgments

This work is framed within the European Commission VI Framework IP Project SeCSE (Service Centric System Engineering) (*http://secse.eng.it*), Contract No. 511680.

References

1. R. Aggarwal, K. Verma, J. Miller, and W. Milnor. Constraint driven web service composition in METEOR-S. In *Proc. IEEE International Conference on Services Computing (SCC'04)*, pages 23–30, Shanghai, China, Sept. 2004.
2. G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. An Approach for QoS-aware Service Composition based on Genetic Algorithms. In *Proc. of the Genetic and Computation Conference (GECCO'05)*, pages 1069–1075, Washington, USA, June 2005. ACM.
3. G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. QoS-Aware Replanning of Composite Web Services. In *Proc. International Conference on Web Services (ICWS'05)*, pages 121–129, Orlando, FL, Jul. 2005. IEEE.
4. J. Cardoso. *Quality of Service and Semantic Composition of Workflows.* PhD thesis, Univ. of Georgia, 2002.
5. J. Cardoso, A. P. Sheth, J. A. Miller, J. Arnold, and K. J. Kochut. Modeling quality of service for workflows and web service processes. *Web Semantics Journal: Science, Services and Agents on the World Wide Web Journal*, 1(3):281–308, 2004.
6. H. Ludwig, A. Keller, A. Dan, R. King, and R. Franck. Web Service Level Agreement (WSLA) language specification. *http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf.*
7. M. Serhani, R. Dssouli, A. Hafid, and H. Sahraoui. A QoS broker based architecture for efficient web services selection. In *Proc. International Conference on Web Services (ICWS'05)*, pages 113–120, Orlando, FL, Jul. 2005. IEEE.
8. K. Verma, K. Gomadam, J. Lathem, A. P. Sheth, and J. A. Miller. Semantics enabled dynamic process configuration. Technical report, LDIS, University of Georgia, 2006.
9. J. Warmer and A. Kleppe. *The Object Constraint Language.* AW, 1999.
10. T. Yu and K. Lin. Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints. In *Proc. 3rd International Conference on Service Oriented Computing (ICSOC'05)*, pages 130–143, Amsterdam, The Netherlands, December 2005. Springer.
11. L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, May 2004.