Service QoS composition at the level of part names

Marco Aiello,[1] Florian Rosenberg,[1] Christian Platzer,[1] Agata Ciabattoni,[1,2] and Schahram Dustdar[1]

[1] VitaLab, Distributed Systems Group, Information Systems Institute
Vienna University of Technology
1040 Vienna, Argentinierstrasse 8/184-1
Austria
{aiellom,florian,christian,dustdar}@infosys.tuwien.ac.at

[2] Institute für Diskrete Mathematik und Geometrie
Vienna University of Technology
1040 Vienna, Wiedner Hauptstrasse 8-10
Austria
agata@logic.at

Abstract. The cornerstone for the success of Service-Oriented Computing lies in its promise to allow fast and easy composition of services to create added-value applications. Compositions need to be described in terms of their desired functional properties, but the non-functional properties are of paramount importance as well. Inspired by the Web Service challenge we propose a new model for describing the Quality of Service (QoS) of a composition which considers the information flow and describes basic service qualities at the granularity level of service part names, that is, operations comprised in service invocation/response messages. In this initial investigation, we overview a number of formal methods techniques that allow to reason with QoS composition based on the proposed model, and propose an algorithm for determining the QoS of a composition given the QoS associated with the individual services.

Keywords: Service-Oriented Computing, Web Services, Service Composition, Quality of Service

1 Introduction

Service-Oriented Computing (SOC) is an emerging computing paradigm for building distributed information systems in which the concepts of distribution, openness, asynchronous messaging and loose coupling take a leading role. In this context, applications are built out of individual services that expose functionalities by publishing their interfaces into appropriate repositories, abstracting entirely from the underlying implementation. Published interfaces may be searched by other services or users and subsequently be invoked. The interest

in Service-Oriented Computing is a consequence of the shift from a vision of a Web based on the presentation of information to a vision of the Web as computational infrastructure, where systems and services can interact in order to fulfill user's requests. Web Services (WS), the best-known example, are the realization of service-oriented systems based on open standards and infrastructures, extending the XML syntax [4].

Web Service technology is being increasingly adopted. Particularly successful are the protocols for the transport of messages (SOAP)[3] and for the description of basic service operations (the Web Service Description Language WSDL).[4] The latter protocol describes messages to be exchanged with a remote Web Service. Exchanged messages are a set of *part names*, that is, operation name and input and output types. The description of functional Web Service properties is thus covered by the WSDL standard. But functional properties are not enough. In fact, non-functional properties of any information systems are as important as the functional ones. Having to wait too long for the output of a system can make it as useless as not having the system at all. This is even more true when considering loosely coupled distributed systems such as those designed following the SOC paradigm.

Quality of Service is the set of properties of a service which have to do with 'how' a service is delivered rather than 'what' is delivered. There is no shared agreement on what QoS is and what is not, but generally properties such as response time, latency, availability, and costs are regarded as QoS. Classifications of QoS features in the context of Web Services have been proposed by several authors [13, 10, 17]. For instance, Ran [15] proposed a QoS model and a UDDI extension for associating QoS to a specific Web Service. An approach for defining QoS requirements is QML [9]: a language for QoS description using XML. QoS aspects are qualified by characteristics as direction and value type. A set of measures for reliability and performance are proposed. Atzeni and Lioy [5] overview security system assessment methods and metrics. A number of approaches to QoS description of services rely on extensions of WSDL, e.g., [10, 18]. The main idea is simple: provide syntax to define terms which refer to non-functional properties of operations. Given such description, one can then build a framework for the dynamic selection of Web Services based on QoS requirements. In [20, 1], the description of elementary service qualities as a quality vector each component of which is a quality parameter for the service is proposed. In [11] Lin, Xie, Guo and Wang use fuzzy logic techniques to handle QoS requirements. The description of QoS of services can also be the object of the negotiation of services in long running-transactions or repeated interactions. QoS become then the object of Service-Level Agreement, see e.g. [12]. We investigated the use of formal methods to describe service level agreements in [2].

In this paper we focus on the composition of services especially considering Quality of Service aspects. A service composition is a set of services together with rules specifying how the various service work together to perform a common

[3] `http://www.w3.org/TR/soap`
[4] `http://www.w3.org/TR/wsdl`

task. There are various issues related to QoS composition. One could have a design of a composition with information regarding QoS of individual elements and wish to know the resulting QoS of the composition. One could have an abstract composition and might need to decide which services to select when implementing the composition in order to fulfill some QoS desiderata, e.g., [14]. In [20, 19], the authors propose a QoS model and a middleware approach for dynamic QoS-driven service composition. They investigate a global planning approach to determine optimal service execution plans for composite service based on QoS criteria. Another interesting question is that of determining the QoS of a composition given basic QoS information of single service operations. In [6], a method is proposed to assess the QoS of a workflow, given the QoS of the individual tasks of the workflow. The methodology consists of a set of rewrite rules for the workflow aiming at arriving at the description of the QoS of the whole workflow.

In this paper, we consider the problem of QoS composition from a different perspective. Instead of resorting to a state based representation giving emphasis to tasks and the flow of control, as e.g. in [6], we take a stateless representation of composition, with individual services as elementary components, and WSDL message part names to represent the data flow. This choice is motivated by the Web Service challenge (see `http://www.comp.hkbu.edu.hk/~ctr/wschallenge/` and `http://insel.flp.cs.tu-berlin.de/wsc06/`) that consists in finding a composition of services which satisfies a given query. The granularity level of the query is at the level of message part names and the composition is modeled as a multigraph of services with part names as edges. In the present work, we generalize the simple model of the Web Service challenge to include Quality of Service attributes, but also to allow defining different patterns in the composition by introducing input service expressions, built using logical operators. The resulting model turns out to be a compact form in which services have a central role and one can appreciate the message exchanged among services.

The rest of the paper is organized as follows. In Section 2, we introduce a simple running example of an application to know the temperature at a given location based on several services. In Section 3, we introduce the QoS model. Formal methods to reason about the QoS of the composition are discussed in Section 4, where we also give an algorithm for establishing the QoS of a composition. Concluding remarks and open research issues are summarized in Section 5.

2 A service composition example

Suppose one wants to build an application for knowing the temperature at a given location. The application should be built using existing services. The non-functional requirements of the application consider the response time and cost of each run of the system. A design of the application is having a program invoking three services: Google to find out the longitude and latitude of the desired loca-

tion,[5] a weather service to find the temperature,[6] and a temperature converter for having the temperature in either Fahrenheit or Celsius.[7] In addition, some processing will be done internally, e.g., extracting the coordinate information from Google result snippets. The example services should be considered only as motivation for the present work, for the ease of presentation we take the liberty of simplifying part names and messages of the services. We also assume that part names of services match, e.g., the output name of Google matches the input part name of Weather.org, even though this is not true in practice. Matching can be achieved resorting to semantic web, or more generally, ontology techniques (see for instance [1]) or by syntactic matching (see for instance [8]).

The input of the application is a text string identifying the location and a date. The output is a temperature in Celsius. Next, we consider how this simple example is formally modeled taking into account both the functional and the non-functional properties of the services, of the composition and the query.

3 Service Model

Web Services standards originated from the industrial need for loosely coupled interprocess communication, there is very little formality beyond the mere XML schema definitions. Here we provide a formalization which allows us to represent both the functional and non-functional properties of services, of service compositions and of queries. Let us begin by the domain of our information system.

Definition 1 (functional service model). *A* functional service model *is a tuple* $\langle S, P, M, in, out \rangle$ *defined in the following way:*

- S *is a set of services,*
- P *is a set of part names,*
- $M \in \mathcal{P}(P)$ *is a set of messages consisting of part names*
- in *is a function* $S \rightarrow \mathcal{P}(P)$*, the set of input part names of a service,*
- out *is a function* $S \rightarrow \mathcal{P}(P)$*, the set of output part names of a service.*

By this definition, a *service* is thus a collection of input and output part names grouped into messages. In the present treatment, we do not consider part types and we use the message information to classify part names into input and output for the various services.

Example 1. Considering the weather example of Section 2, S consists of the services $\{Google, Weather.org, ITempConverter\}$, P consists of many part names, such as the following ones:

```
Google:
  <xsd:element name="searchQuery" type="xsd:string"/>
  <xsd:element name="searchTime" type="xsd:double"/>
```

[5] http://www.Google.com/apis/

[6] http://www.weather.gov/forecasts/xml/DWMLgen/wsdl/ndfdXML.wsdl

[7] http://developerdays.com/cgi-bin/tempconverter.exe/soap/ITempConverter

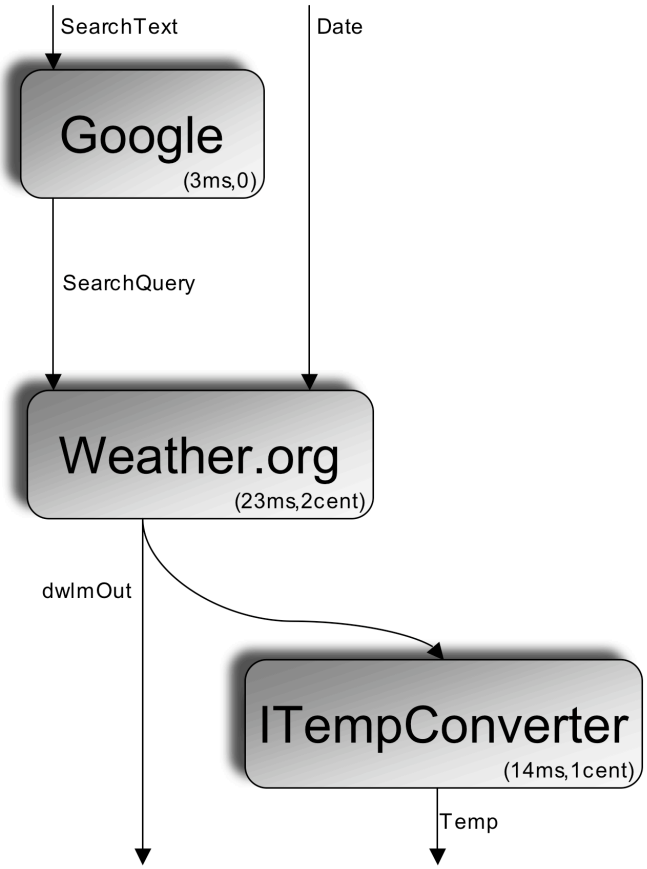

Fig. 1. The weather example modeled.

```
Weather.org:
  <xsd:element name="temp" type="xsd:boolean"/>
  <part name="dwmlOut" type="xsd:string" />

ITempConverter:
  <part name="temp" type="xs:int"/>
```

an example of a message in M is given by the message of Google consisting of an input and an output message

```
  <operation name="doGoogleSearch">
    <input message="typens:doGoogleSearch"/>
```

```
    <output message="typens:doGoogleSearchResponse"/>
  </operation>
```

Finally, an example of an output function for the Google Web Service is, omitting the XML syntactic sugar, *out*(Google)={searchQuery, searchTime}.

Let us now consider the non-functional properties by introducing our QoS model.

Definition 2 (QoS service model). *A* Quality of Service model *is an ordered set of groupoids* $\langle (G_i, \check{\star}_i \hat{\star}_i)_{i=1,\ldots,n} \rangle$*, where each groupoid* i *consist of a set* G_i *with two operations* $\hat{\star}_i$ *and* $\check{\star}_i$*. A* QoS element *with respect to a QoS model is a vector* $\langle q_1 \ldots q_n \rangle$ *were* $q_i \in G_i$*, for each* $i = 1, \ldots, n$*. We denote by* $\check{\star}(q_a, q_b)$ *and* $\hat{\star}(q_a, q_b)$ *the componentwise operations* $(q_{a_1} \check{\star}_1 q_{b_1} \ldots q_{a_n} \check{\star}_n q_{b_n})$ *and* $(q_{a_1} \hat{\star}_1 q_{b_1} \ldots q_{a_n} \hat{\star}_n q_{b_n})$ *among two services* a *and* b *with QoS elements* $q_a = \langle q_{a_1} \ldots q_{a_n} \rangle$ *and* $q_b = \langle q_{b_1} \ldots q_{b_n} \rangle$*.*

Notice that each groupoid models a QoS requirement and the groupoid operations, interpreting the operators in Definition 4, will be used to compute the QoS of a given composition.

Example 2. The weather example presented in Section 2 considers two QoS requirements. One tied to execution time and one to costs. Therefore, the QoS service model consists of two groupoids, e.g., the real numbers with the addition and the average for considering time and the integers with addition and max for the cost. Then we have that any part name associated with Google has a quality cost which is zero, while an execution time which is in the range of the few seconds. The latter can be modeled in various ways. One can take the average of the execution times experienced in the past, one can consider the value returned by Google itself as output in searchTime for a given request. One may even look at a finer granularity of the execution time as we do in [16]. The choice is not relevant for the present treatment.

Having defined what a service is from a functional and a from a non-functional point of view, let us consider collections of services populating the same network which can be invoked as parts of a same composition process. Such a composition can be the result of a design process or of a search to satisfy a service query. Let us define the latter concept formally.

Definition 3 (service query). *A* service query *over a set of services* S *is an expression of the form* i^*, o^+ *where* $i \in P$ *are the optional input query part names,* $o \in P$ *are the query output part names, and* $^*, ^+$ *are the usual Kleene string operators.*

Example 3. The service query `SearchText Date, Temp` means that the requester provides a text and a date, and desires to get a temperature.

Definition 4 (input service expressions). *An* input service expression *associated to a service* S_1 *is a string built over the input part names of* S_1 *(called* atoms*) using the binary, associative, and commutative operators* $\bar{\wedge}$ *and* $\bar{\vee}$ *and the auxiliary symbols* $(,)$*.*

Example 4. An input service expression associated to a "transform address into zip code" service which has in ={address, city, zip_code}, out ={zip_code,time} is (address$\bar{\wedge}$city)$\bar{\vee}$zip_code whose intended semantics is that either a zip code or an address **and** a city are provided.

We are now in the position of defining a service composition.

Definition 5 (service composition). *A* service composition *over a service collection* $C = \langle S, P, M, in, out\rangle$ *and QoS model* $\langle (G_i, \bar{\star}_i \hat{\star}_i)_{i=1,\ldots,n}\rangle$*, is a labeled multigraph* $\langle V, E, ExpI, Q_V\rangle$ *with the following properties:*

1. *each element* $v \in V$*, is either in* S *or* $\exists\ v' \in V \cap S$ *such that the services* v *and* v' *differ only for their names.*
2. $E \subseteq V \times V \times P$*, and* $e = \langle v_1, v_2, p\rangle \in E$ *if* $out(v_1) = in(v_2) = p \in P$
3. $ExpI$ *is a function associating to each element* $v \in V$ *an input service expression associated to* v*.*
4. Q_V *is a function associating to each element* $v \in V$ *a QoS element.*

Condition 1. in the above definition says that multiple occurrences of a service in the multigraph are identified using different node names. Condition 2. says that there is an edge in the graph connecting two services only if a part name is output and input of the two services, respectively. Condition 3. and 4. specify the labels assigned to each node v: an input expression ($ExpI(v)$) and an element of QoS ($Q_V(v)$) that is, the quality of the individual service.

Example 5. Following the above definition, the composition presented in Section 2 is then modeled as shown in Figure 1. Where the query is `SearchText Date, Temp`. Consider the service Weather.org: its associated input expression can be SearchQuery$\bar{\wedge}$Date "meaning" that both a SearchQuery and a Date must be provided while (23ms,2cent) stands for its QoS values of time and cost.

4 Model inspection, checking, construction

Having a formal model of services and their compositions from a functional and non-functional point of view enables the use of a number of formal methods techniques to reason about services. The main methods to be used range from the simple model inspection to determine the QoS of a given composition, to the model checking of a composition, up to the more complex task of model construction. Figure 2 summarizes the most interesting methods and the tasks they address. In the present treatment we take a closer look at the first one, that is, the model inspection for determining the QoS of a given composition.

In [3], we provided algorithms for dealing with the model construction problem where we do not consider input expressions for QoS. The same problem is solved using a partial order planner in [7].

method	**input**	**task**	**output**
model inspection	a composition and a query	know the QoS for the query	an element of the QoS model
model checking	a composition, a query, and a QoS property	find if the query satisfies the QoS	yes/no
(directed) model checking	a composition, a query, and a QoS property	find, if it exists, the proof which is optimal w.r.t. QoS	optimal proof
model construction	a query, a functional and QoS service models	create a composition	the composition, if it exists, satisfying the query

Fig. 2. Methods to reason about QoS service composition.

4.1 Modeling at the level of part names

Given a composition of services (that is, a multigraph like the one in Figure 1 together with input, and QoS values) and a query stating which part names are available and which are the desired ones, we want to arrive at the determination of the QoS of the composition for the given query. But first we need to lift the Q_V function, that associates qualities of services with services (nodes v in the labeled multigraph) in the composition, to input service expressions. We do so using the following recursive definition.

Definition 6. *(input expression QoS) Given a service composition* $\langle V, E, ExpI, Q_V \rangle$*, let* $v \in V$ *and* $e, e_1, e_2 \in ExpI(v)$*, then the* input expression QoS function Q *over an input service expression* e *is defined in the following way:*

- *if* e *is an atom,* $Q(e) = Q_V(w)$*, where* $\langle w, v, e \rangle$ *is in* E*;*
- $Q(e_1 \bar{\wedge} e_2) = \hat{\star}(Q(e_1), Q(e_2))$ *where* e_1, e_2 *are input expressions and* $\star$ *are the first operators of the respective QoS groupoids;*
- $Q(e_1 \bar{\vee} e_2) = \check{\star}(Q(e_1), Q(e_2))$ *where* e_1, e_2 *are input expressions and* $\hat{\star}$ *are the second operators of the respective QoS groupoids.*

We remark that the $\hat{\star}$ and $\check{\star}$ operators are chosen when designing the composition.

Example 6. If we are interested in QoS time, then it could be modeled by a groupoid whose universe is the set of real numbers and whose operations $\hat{\star}$ and

$\star$ could be the addition and the max function. The operations's choice depends on the considered web service composition and on the goal of the QoS model as defined by the composition designer or user. E.g., addition and max allow both sequential and parallel arcs to be modeled in the service composition graph. On the other hand, when parallel arcs do not occur in the service composition graph and we are interested in the average QoS of the composition, then the function max could be replaced by the function average.

Notice that in our model, the information on how services relate/interact are contained both in the arcs and in the input service expressions associated to nodes of the labeled multigraphs. This renders the modeling of composition provided a more compact and flexible form for representing Web Service compositions than, e.g., workflows. For instance, the sequential composition at the task level of Fig. 3 (assume the operations between S_1 and S_2 consist of the three part names p_a, p_b and p_c and the considered QoS is time) can be represented by the labeled composition multigraph of Fig. 4. in which $ExpI(S_2) = p_a \bar{\wedge} p_b \bar{\wedge} p_c$

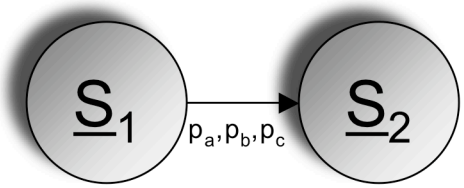

Fig. 3. Sequential flow.

and the operator $\bar{\wedge}$ is interpreted as real numbers addition. Taking however $ExpI(S_2) = p_a \bar{\vee} p_b \bar{\vee} p_c$, where $\bar{\vee}$ is interpreted as the maximum between real numbers, the composition multigraph of Fig. 4 . would then correspond to the parallel composition at the task level of Fig. 5. Therefore, by changing the input service expressions associated to S_2 (while the interpretations of $\bar{\wedge}$ and $\bar{\vee}$ remain the same), the composition multigraph of Fig. 4 would correspond to 2^3 different workflows.

Of course, there are other differences among the modeling we propose at the part name level and workflows beside the compact representation of the former with respect to the latter. The most notable differences include: stateless vs. statefull representation and data centered representation vs. control flow representation, respectively.

4.2 Model inspection

In the following we assume there are no loops and that the compositions are correctly designed with respect to the queries. Relaxing the former assumption

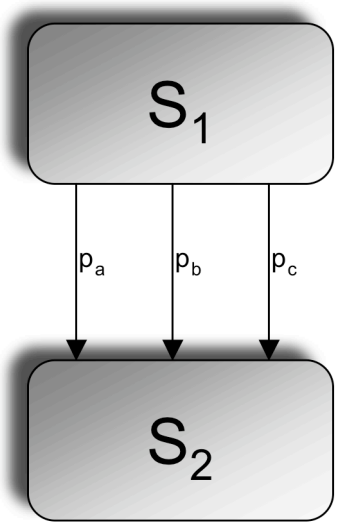

Fig. 4. Composition at the part name level.

requires appropriate algorithms in the spirit of [6], while relaxing the latter assumption brings us to the terrain of model checking, rather than model inspection.

Algorithm 1 Model Inspection(composition $\langle V, E, ExpI, Q_V \rangle$, query i, o)

$V = V \bigcup \{$query Qu_I, Qu_O nodes created using $i, o\}$
active parts $= i$
QoS associated with Qu_I set to the default value
loop
  consider a node $v \in V$ such that $in(v) \in$active parts
  active parts = active parts $\bigcup out(v)$
  $Q(v) = \hat{\star}(Q(v), Q(ExpI(v)))$ according to Definition 6
  if $v = Qu_O$ **return** $Q(Qu_O)$
end loop

The algorithm (Algorithm 1) for model inspection works by traversing the composition graph and computing the QoS of the composition. The algorithm takes a composition graph and a query. It uses the query for determining the set of initial active parts and builds two extra nodes to represent the query input Q_I and output Q_O. Active parts are the messages which are available for the composition. The vector QoS keeps the value of the QoS during the computation and is initially set to the default values (for instance cost is set to 0). The loop

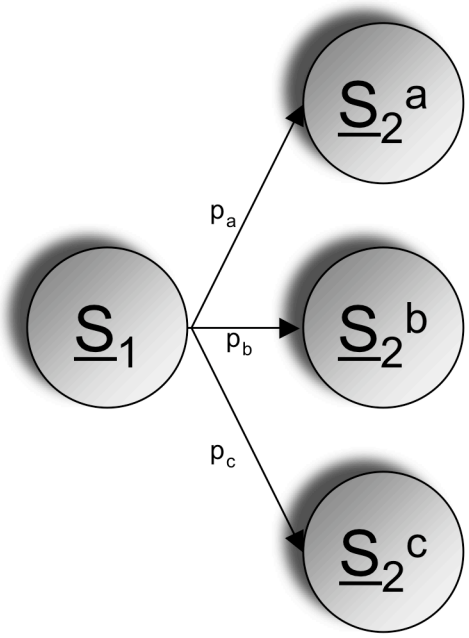

Fig. 5. Parallel flow.

of the algorithm takes nondeterministically a node for which all input parts are active. Given the assumption of correct design there is always such a node, or we have reached the end of the computation. Then the output part of the considered node are added to the set of active parts. We are now in the position of computing the new QoS for the considered node. The computation of the service QoS in the given composition is performed by computing the QoS of its input expression and 'adding' $\hat{\star}$ the QoS of the service. Of the two groupoid operation sets $\hat{\star}$ and $\check{\star}$, the former is in the algorithm as this is the one which should model the logical and, i.e., the addition of the quality of service computed so far and the quality of service of the specific service. Given the absence of loops we notice that the non-deterministic choice of a node does not affect the correctness of the algorithm. Finally, if the node considered was the final node of the composition we exit the loop returning the computed QoS.

4.3 A run on the weather example

Let us consider again the weather example of Section 2, shown in Figure 1, and apply Algorithm 1. We start by setting the active parts to the query SearchText and Date, adding the node Q_I to which we associate the default quality of service of (0,0): no time and no costs. We also add the node Q_O to represent the end

of the query which has as input parts the queried Temp, its input expression is simply $Temp$. Then the loop begins.

At the first iteration we can only consider the Google service. In fact, its input part names are all active, on the other hand Weather.org has one input part name active (Date) but not the other one (SearchQuery). We then add the output part (SearchQuery) of Google to the active part names and update its quality of service. The quality of service associated with this service was (3,0)—it takes Google 3 milliseconds and it is for free—which is combined with the evaluation of the input expression of SearchText which is (0,0). In this case, the quality of service does not change.

At the following iteration we can choose the Weather.org service. We add its output part to the active part names and then we compute the quality of service for its two inputs. We have (3,0) and (0,0), respectively. Supposing that the input expression is SearchQuery$\bar{\wedge}$Date, that $\hat{\star}$ is modeled as real numbers addition and integer addition, and that its QoS is (23,2), then we update the QoS of Weather org with (26,2). At the final iteration iTempConverter is chosen yielding a final QoS associated with it of (40,3). We then conclude that the QoS of the composition is 40 milliseconds and 3 cents. Again these could be minimum, maximal, average values or something else, depending on the choice made in the composition design.

5 Concluding remarks

We have presented preliminary work aimed at modeling Web Service compositions from a functional and non-functional point of view at the granularity level of the part names. Following this modeling, we overview a number of formal methods techniques that allow to reason with QoS composition based on the proposed model, and propose an algorithm for determining the QoS of a given composition given the QoS associated with the individual services.

In this initial work, we made a number of simplifying assumptions which we will remove in future work. In particular, we have not considered loops in the compositions while these could be present and need to be modeled. We have not presented output expressions (the natural counterpart of input expressions for services), and we have not considered limitations on the use of part names (for instance, one could impose that a part name is used only once by any service). Furthermore, we have only provided an algorithm for the case of model inspection, leaving open the challenge of finding algorithms for model checking and model constructions.

Acknowledgments

We thank Ganna Frankova for comments on a previous version of this paper and discussion on related work.

References

1. R. Aggarwal, K. Verma, J. Miller, and W. Milnor. Constraint Driven Web Service Composition in METEOR-S. In *Proceedings of the 2004 IEEE International Conference on Services Computing*, Shanghai, China, September 2004.
2. M. Aiello, G. Frankova, and D. Malfatti. What's in an agreement? A formal analysis and an extension of WS-Agreement. In *Int. Conf. on Service-Oriented Computing (ICSOC 2005)*, 2005.
3. M. Aiello, C. Platzer, F. Rosenberg, H. Tran, M. Vasko, and S. Dustdar. Web service indexing for efficient retrieval and composition. In *In Joint 8th IEEE Conference on E-Commerce Technology (CEC'06) and the 3rd IEEE Conference on Enterprise Computing, E-Commerce and E-Services*. IEEE Computer, 2006. To appear.
4. G. Alonso, F. Casati, H Kuno, and V. Machiraju. *Web Services*. Springer-Verlag, 2004.
5. A. Atzeni and A. Lioy. Why to Adopt a Security Metric? A Brief Survey. In *Proceedings of the First Workshop on Quality of Protection*, Milan, Italy, September 2005.
6. J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and web service processes. *Journal of Web Semantics*, 1(3):281–308, 2004.
7. J. Dorn, P. Hrastnik, and A. Rainer. Web service discovery and composition for virtual enterprises. *International Journal of Web Services Research*, 2006. To appear.
8. I. Elgedawy, Z. Tari, and M. Winikoff. Exact functional context matching for web services. In M. Aiello, M. Aoyama, F. Curbera, and M. P. Papazoglou, editors, *Int. Conf. on Service-Oriented Computing (ICSOC 2004)*, pages 143–152. ACM, 2004.
9. S. Frølund and J. Koistinen. Quality-of-Service Specification in Distributed Object Systems. *Distributed Systems Engineering*, 5(4):179–202, December 1998.
10. D. Gouscos, M. Kalikakis, and P. Georgiadis. An approach to modeling web service QoS and provision price. In *1st Web Services Quality Workshop (WQW2003) at WISE*, 2003.
11. M. Lin, J. Xie, H. Guo, and H. Wang. Solving QoS-Driven Web Service Dynamic Composition as Fuzzy Constraint Satisfaction. In *Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service*, Hong Kong, China, March-April 2005.
12. H. Ludwig. Web services QoS: External SLAs and internal policies or: How do we deliver what we promise? In *1st Web Services Quality Workshop (WQW2003) at WISE*, 2003.
13. D.A. Menasce. QoS issues in Web services. *IEEE Internet Computing*, 6(6):72–75, November/December 2002.
14. D.A. Menasce. Composing Web Services: A QoS View. *IEEE Internet Computing*, 8(6):88–90, November/December 2004.
15. S. Ran. A model for web services discovery with QoS. *SIGecom Exchanges*, 4(1):1–10, 2003.
16. F. Rosenberg, C. Platzer, and S. Dustdar. Bootstrapping performance and dependability attributes of web services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'06)*. IEEE Computer, 2006. To appear.

17. M. Tian, A. Gramm, T. Naumowicz, H. Ritter, and J. Schiller. A concept for QoS integration in web services. In *1st Web Services Quality Workshop (WQW2003) at WISE*, 2003.
18. M. Tian, A. Gramm, H. Ritter, and J. Schiller. Efficient Selection and Monitoring of QoS-aware Web Services with the WS-QoS Framework. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*, Beijing, China, September 2004.
19. L. Zeng, B. Benatallah, Ngu; A.H.H., 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.
20. L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q.Z. Sheng. Quality driven web services composition. In *Proceedings of the 12th International Conference on World Wide Web (WWW'03)*, pages 411–421, New York, NY, USA, 2003. ACM Press.