A Model-Based Approach for the Pragmatic Deployment of Service Choreographies

Raphael Gomes[1,2](✉), Júnio Lima[1], Fábio Costa[1], Ricardo da Rocha[1], and Nikolaos Georgantas[2]

[1] Instituto de Informática, Universidade Federal de Goiás, Goiânia, Brazil
raphael.gomes@ifg.edu.br, junio.lima@ifgoiano.edu.br, {fmc,ricardo}@inf.ufg.br

[2] MiMove Team Inria Paris, Rocquencourt, France
nikolaos.georgantas@inria.fr

Abstract. The development of applications using service choreographies is becoming one of the *de facto* standards for the Future Internet. However QoS-aware management of service compositions is usually performed without considering service sharing. This simplifying assumption makes choreography deployment less feasible in real scenarios, in which a single service is typically shared in many scenarios. In this paper we discuss the problem of managing multiple choreographies in multi-cloud environments and we advocate that sharing-aware deployment is a more effective and resource-efficient approach. We propose a model for the combined deployment of multiple choreographies on top of a shared set of services, and we further investigate the problem through experiments.

1 Introduction

Among its new features, the Future Internet is characterized by the evolution from content sharing to service sharing. In this new scenario, mainly facilitated by the adoption of cloud technologies, software modules of different complexities are provided on top of virtualized servers and consumed via the Internet [1].

Keeping centralized coordinators for these new types of applications is unfeasible due to requirements like fault tolerance, availability, heterogeneity and adaptability. For this reason, a promising solution is the use of decentralized and distributed services through choreographies. Choreographies are service compositions that implement distributed business processes in order to reduce the number of exchanged control messages and distribute business logic, without the need for centralized coordinators [2]. Building a choreography is usually a two-step task [3]. Firstly, the functionalities required from the participating services, i.e., their operations, are identified. Secondly, for each operation an appropriate implementation is selected and bound to it. The activity of performing the interactions and getting the expected results is named choreography enactment.

In most cases, service selection and choreography enactment are not based solely on functional criteria. Instead, they aim to satisfy non-functional requirements as well, in terms of Quality of Service (QoS) properties, which in turn

A. Celesti and P. Leitner (Eds.): ESOCC 2015 Workshops, CCIS 567, pp. 153–165, 2016.
DOI: 10.1007/978-3-319-33313-7_11

poses many challenges. For implementation selection, the growing number of alternative web services that provide the same functionality but differ in quality parameters makes service selection an NP-hard optimization problem [4]. On the other hand, along with choreography enactment, resource allocation plays an important role in QoS since almost all non-functional requirements are related to the resources used to deploy the services.

The problem of QoS-aware choreography enactment is usually solved using variations of the Knapsack Problem [5]. However, all these solutions assume that there are no conflicts between the services that are part of a choreography, such as heterogeneous communication protocols. They also do not take into account the fact that a given service may be part of more than one choreography, which in turn means that requests for the same operation of a service may come concurrently from different sources and with different QoS requirements. As a consequence, service implementation selection and resource management typically take into account QoS requirements that are specific to a single choreography. This is far from ideal, given the combined QoS-related constraints that arise from the sharing of services among multiple and diverse choreographies.

We argue that a pragmatic view of choreography deployment, based on service sharing and on management on a per operation basis, represents a more realistic perspective since each service can have different QoS requirements for the same operation depending on which choreography is generating calls to it. With this in mind, we propose a model-based approach that encompasses both choreography deployment and resource management. We first formalize a working terminology (Sect. 2) and discuss the effects of service sharing in choreographies based on some experimental results (Sect. 3). We then examine related work (Sect. 4), and propose a formal model (Sect. 5) to represent service choreographies taking into account a global view of service utilization and the associated non-functional requirements. We aim to use this model as part of a adaptive approach for choreography enactment, which is discussed in Sect. 6. Finally, Sect. 7 presents some final remarks.

2 Terminology

In our work, an **application** is a web-based computer program designed for a specific use, such as an application for setting up a doctor's appointment in the public health system. An application is composed using one or more services. A **service**, in turn, is an independent software component that executes one or more operations. An **operation** defines some action performed by the service. It requires some amount of computing power to be processed.

The composition of several services by means of their provided operations forms a **choreography**. As pointed out before, a choreography is a form of service composition where the interaction protocol (among services) is defined in a global way using a peer-to-peer approach. The services that compose a choreography can be described in an abstract way by means of the expected role that each service plays in the interaction. Such abstract services can be realized using concrete entities, i.e., by identifying a target implementation for each service. Abstracting services is particularly important in multi-cloud environments, as some implementations can be specific to a cloud provider/technology. In such environments, composition deployment may become overly constrained if concrete services are used instead to specify a choreography.

User refers to the person(s) responsible for application composition and administration, which includes service selection and resource allocation. Both tasks must be performed with the aim of satisfying the functional and non-functional requirements of clients. Another task performed by the user is to manage adaption of service selection and resource allocation in the face of changes in the system's conditions and in the client's expectations. The **client** in turn is an entity that interacts with the application. It mainly refers to the end-user.

Finally, a **system** is a set of interacting or interdependent components forming an integrated whole. We use this term to refer to the set of managed applications, together with the components required to implement our approach.

3 The Effect of Service Sharing

Nowadays applications are developed mainly by means of preexisting service compositions. QoS management in this composite scenario is even more difficult if we consider that a service may be used by many applications at the same time. For instance, a maps service can be used in applications such as driving directions guides, picture location tagging, and partner matching by location. For each of these applications, the service may have different QoS requirements.

As illustrated in Fig. 1, this scenario is equivalent to managing a dancer participating in a music mash-up choreography: she must be able to properly handle the multiple requests and perform the different dance rhythms with an expected quality. In the same sense, for a given application (which is analogous to the mash-up in our metaphor) there can be several choreographies (rhythms in our metaphor), with the same services (analogous to dancers) being shared among them with different QoS requirements. Therefore, it is not possible to manage services without considering all the choreographies in which they participate. To achieve this goal, we need to act upon resource allocation, as the majority of non-functional properties are related to the use of resources.

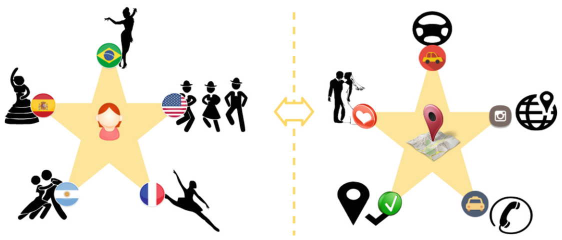


Fig. 1. Mash-up metaphor.

Thus, in our approach, QoS specification is done at two different levels: service and choreography. The first level specifies requirements regarding a specific service/operation, without taking into account end-to-end QoS. The second level concerns the quality of the choreography as a whole (i.e., end-to-end) and is evaluated in terms of the composition of all participating services.

We have performed a set of experiments to demonstrate the effect of service sharing on choreography QoS. Thereby, we aim to demonstrate the need for global resource management across choreographies for the effective satisfaction of QoS requirements. The experiments show that service concurrency does not make QoS satisfaction unfeasible, provided that proper resource management is performed. We present our analysis results in the following.

3.1 Evaluating the Effects of Service Sharing

Our analysis of the effects of service sharing is based on queueing theory. For this purpose, we used JINQS [6], a library for simulating multiclass queuing networks. We evaluated the execution of different choreographies composed by non-intersecting service sets, as well as choreographies that use shared services. In our experiment we generated random choreography topologies with sequential and branching control flow patterns. For simplicity, we assume that each service provides only one operation, whose processing time follows an exponential distribution with rate parameter μ taking values between 2 and 200 (meaning that the mean processing time is between 0.5 and 0.005 time units). By putting together generated choreography topologies, we create sets of choreographies. Considering the choreography topologies of a set separately and in combination, we model them as queueing networks and simulate them on JINQS. To each choreography we apply an external input load following a Poisson distribution with rate parameter $\lambda = 50$ requests per time unit.

We simulated different levels of service sharing among the choreographies, varying from 0 % (no sharing) to 100 % (all services are shared). With this in mind, we generated a service base of available services, from which we randomly selected 10 services each time in order to compose the choreographies, according to the chosen *service sharing level*. Note that this parameter only indicates the probability of having a specific number of services shared among the choreographies (it does not mean that all services are necessarily shared among all choreographies). We also analyzed different numbers of choreographies combined together, with 2, 4, 8 and 16 choreographies being enacted at same time.

As target metrics we first measured the number of served (completed) requests and the average response time (RT). The results are presented in Tables 1 and 2, which show the mean of the differences in the two metrics for running the choreographies in isolation and in combination, with a confidence interval of 95 %. Positive values indicate loss of QoS when executing choreographies in combination. Hence, negative values indicate better QoS. As expected, service sharing causes loss of QoS since both metrics are worse when we execute a higher number of choreographies concurrently. Another interesting result is that the number of served requests is less influenced by changes in the level of

Table 1. Mean difference (%) between the numbers of completed requests when running the choreographies in isolation and in combination.

Sharing/# Chor.	2	4	8	16
0 %	-0.02 ± 0.11	-0.04 ± 0.06	0.03 ± 0.05	-0.02 ± 0.03
25 %	-0.11 ± 0.11	-0.04 ± 0.07	0.00 ± 0.05	56.19 ± 0.05
50 %	-0.02 ± 0.12	-0.01 ± 0.08	41.32 ± 0.08	101.06 ± 0.05
75 %	0.01 ± 0.11	0.04 ± 0.07	41.34 ± 0.07	101.03 ± 0.05
100 %	-0.08 ± 0.09	-0.02 ± 0.07	63.20 ± 0.07	117.47 ± 0.06

Table 2. Mean difference (%) between the response times when running the choreographies in isolation and in combination.

Sharing/# Chor.	2	4	8	16
0 %	-9.88 ± 0.05	-12.20 ± 0.03	-13.32 ± 0.03	-13.42 ± 0.02
25 %	-7.68 ± 0.06	-4.31 ± 0.04	42.11 ± 0.18	199.94 ± 0.00
50 %	-2.80 ± 0.09	22.12 ± 0.11	199.93 ± 0.00	199.96 ± 0.00
75 %	3.60 ± 0.07	43.37 ± 0.11	199.93 ± 0.00	199.96 ± 0.00
100 %	13.46 ± 0.10	121.74 ± 1.16	199.95 ± 0.00	199.96 ± 0.00

Table 3. Mean difference (%) between the numbers of completed requests when running the choreographies in isolation and in combination (with the addition of more resources when running them in combination).

Sharing/# Chor.	2	4	8	16
0 %	-0.01 ± 0.12	-0.05 ± 0.07	-0.01 ± 0.04	-0.01 ± 0.03
25 %	-0.06 ± 0.10	-0.01 ± 0.06	-0.04 ± 0.05	0.02 ± 0.03
50 %	-0.12 ± 0.11	0.04 ± 0.07	-0.00 ± 0.04	3.77 ± 0.04
75 %	0.11 ± 0.13	0.06 ± 0.07	-0.04 ± 0.05	3.75 ± 0.05
100 %	0.04 ± 0.13	-0.00 ± 0.06	0.00 ± 0.05	27.07 ± 0.06

Table 4. Mean difference (%) between the response times when running the choreographies in isolation and in combination (with the addition of more resources when running them in combination).

Sharing/# Chor.	2	4	8	16
0 %	-9.96 ± 0.05	-12.22 ± 0.04	-13.30 ± 0.03	-13.42 ± 0.02
25 %	-25.68 ± 0.07	-28.79 ± 0.04	-28.61 ± 0.03	-24.13 ± 0.02
50 %	-43.85 ± 0.07	-46.98 ± 0.04	-43.54 ± 0.03	199.34 ± 0.01
75 %	-74.58 ± 0.06	-76.48 ± 0.04	-66.42 ± 0.03	199.36 ± 0.01
100 %	-106.46 ± 0.05	-101.48 ± 0.03	-77.25 ± 0.05	199.89 ± 0.00

sharing, while RT doesn't change significantly as the number of choreographies increases.

We also analyzed the impact of resource allocation. To this end we carried out the same experiment, now increasing resource allocation by a factor of 1 to 3 for combined choreography execution. Tables 3 and 4 show the results. The behavior is similar, although with a smaller difference between separately executing each choreography and executing all of them in combination. This reinforces the motivation for using a more precise resource allocation.

Motivated by these results, our proposal is to automate the management of service selection and resource allocation in multi-cloud environments taking into account service sharing. We propose the representation of services and resources in abstract models which are dynamically interpreted by the system. In the next section we discuss how this aspect is considered in related work. We then present the first step towards defining our approach, which consists in eliciting a formal model to represent combined choreographies.

4 Related Work

A number of research efforts reported in the literature have focused on the problem of providing QoS guarantees for service compositions [3–5]. However, most of these studies focus on service selection for a single composition. To the best of our knowledge, Nguyen et al. [7] carried out one of the first studies to deal with QoS guarantees for multiple inter-related compositions. The authors argue that if a service engages in a number of compositions, there will be a dependency between the levels of QoS that the service can contribute to these compositions. In the approach proposed by Ardagna and Mirandola [8], service composition is carried out based on groups of invocations where multiple requests are generated by multiple users. However, they assume that each service provider has fixed resources, thus not proposing resource adaptability.

Furtado et al. [9] present a middleware to support the enactment of web service choreographies in the cloud. Similarly to our work, resource adaptation is proposed to maintain the expected levels of QoS. However, they do not handle service selection. Huang and Shen [10] propose an approach for the deployment of multiple services in the cloud. They developed two types of graphs to model the communication costs and potential parallelism among the services of different compositions. However, unlike our approach, which focuses on service sharing, they aim at minimizing communication costs and maximizing parallelism.

In contrast, we propose an approach to deal with multiple inter-related service choreographies, taking into account their associated non-functional requirements and a global view of service utilization. We analyze the role each service plays in several choreographies and estimate the amount of resources needed to deploy each service in order to ensure the expected level of QoS.

5 Formal Model for Choreography Deployment

In this section we present a formalization of the problem of combined deployment of multiple choreographies. We focus on non-functional properties, although our formalization can handle functional properties as well. Our representation of choreographies is language-independent but contains the main components of commonly adopted choreography definition languages, such as BPMN2 [11].

The set of available services used to compose choreographies is defined as $\mathcal{S}$, which contains n services $\{s_1, s_2, \ldots, s_n\}$, each represented by a group of operations $\mathcal{O}$. Each operation $o \in \mathcal{O}$ has resource demand d, which represents the amount of resources, e.g., number of CPU cores and their capacity, needed to compute the operation. Moreover, the set of available resources is represented as $\mathcal{V}$, which contains t virtual machine (VM) configurations $\{v_1, v_2, \ldots, v_t\}$. Each resource v has ρ resource units, each one with resource capacity ζ and a cost c for using it for a given time slice. The topology of a choreography can be abstracted using a process graph [12], which is defined as follows:

Definition 1 ***(Predecessor and Successor Nodes).*** *Let N be a set of nodes and $E \subseteq N \times N$ a binary relation over N defining the edges. For each node $n \in N$ we define the set of predecessor nodes $\bullet n = \{x \in N | (x, n) \in E\}$ and the set of successor nodes $n \bullet = \{x \in N | (n, x) \in E\}$.*

Definition 2 ***(Process Graph).*** *A process graph PG consists of a tuple $(b, Z, \mathcal{S}, L, t, E)$ where:*

- *b denotes the start point, $|b \bullet| = 1 \wedge |\bullet b| = 0$.*
- *Z denotes the set of end events, $|Z| \geq 1$ and $\forall z \in Z : |\bullet z| \geq 1 \wedge |z \bullet| = 0$.*
- *$\mathcal{S}$ denotes the set of services, $\forall s \in \mathcal{S} : |\bullet s| = 1 \wedge |s \bullet| = 1$.*
- *L denotes the set of connectors, $\forall l \in L : (|\bullet l| > 1 \wedge |l \bullet| = 1) \vee (|\bullet l| = 1 \wedge |l \bullet| > 1)$.*
- *t is a mapping $t : L \rightarrow \{AND, XOR, OR\}$, which specifies the type of a connector $l \in L$ as either a conjunction (AND), a disjunction (OR) or a mutually exclusive disjunction (XOR).*
- *E is a set of edges that define the flow as a simple and directed graph. Each edge $e \in E$ is a tuple $(\underset{\rightarrow}{e}, \overrightarrow{e}, o)$, where $\underset{\rightarrow}{e} \subseteq (b \cup \mathcal{S} \cup L)$ is the origin of this edge, $\overrightarrow{e} \subseteq (Z \cup \mathcal{S} \cup L)$ is the end of this edge, and o is the operation being requested. If $\overrightarrow{e} \in \{Z \cup L\}$, then o is null. Being a simple graph implies that $\forall n \in (b \cup Z \cup \mathcal{S} \cup L) : (n, n) \notin E$ (no reflexive edges) and that $\forall x, y \in (b \cup Z \cup \mathcal{S} \cup L) : |\{(x, y) | (x, y) \in E\}| = 1$ (no multiple edges).*

In our approach, each expected non-functional requirement is described in terms of a QoS property, which in turn is represented by one or more QoS metrics. These concepts are formalized in the following.

Our representation for QoS metrics is based on Rosario et al. [13]:

Definition 3 ***(QoS Metric).*** *A QoS metric is a tuple $m = (\mathbb{D}, \leq, \oplus, \wedge, \vee, \mathcal{U})$:*

- *$(\mathbb{D}, \leq)$ is a QoS domain with a corresponding set of ordered QoS values.*

- $\oplus : \mathbb{D} \rightarrow \mathbb{D}$ *defines how QoS gets incremented by each new event. It satisfies the following conditions: (i)* $\oplus$ *possesses a neutral element* 0 *satisfying* $\forall l \in \mathbb{D} \Rightarrow l \oplus 0 = 0 \oplus l = l$*; (ii)* $\oplus$ *is monotonic:* $l_1 \leq l'_1$ *and* $l_2 \leq l'_2$ *implies* $(l_1 \oplus l_2) \leq (l'_1 \oplus l'_2)$.
- $(\wedge, \vee)$ *represents the lower and upper lattices, meaning that any* $l \subseteq \mathbb{D}$ *has unique lower and upper values* $(\wedge_l, \vee_l)$*. When taking the best result with respect to the ordering* $\leq$*, the lowest QoS is taken with* $\wedge$*. When synchronizing events, the operator* $\vee$ *takes the worst QoS as per the ordering* $\leq$.
- $\mathcal{U}$ *is a utility function* $\mathcal{U} : (\mathcal{S}, \mathcal{V}) \rightarrow \mathbb{D}$*, that gives the expected QoS value when a service* $s \in \mathcal{S}$ *is deployed on a specific resource* $v \in \mathcal{V}$.

Definition 4 ***(Non-Functional Requirement).*** *A non-functional requirement (NFR) is represented using one of the following tuples:*

(1) (s, o, k, ϕ)*, where* $s \in \mathcal{S}$ *is a service, o is the operation being requested, k is a QoS metric, and* ϕ *is the target average value for this metric* ($\phi \in \mathbb{D}(k)$)*;*
(2) (k, ϕ)*, where k is a QoS metric and* ϕ *is the target average value for this metric, with* $\phi \in \mathbb{D}(k)$*. This tuple is used to represent end-to-end NFRs, which means that the target value must be somehow split among the operations (and respective services) in the possible execution flows.*

To allow QoS-aware choreography enactment, we propose the representation of choreographies and NFRs in a structure called *QoS-Aware Process Graph.*

Definition 5 ***(QoS-Aware Process Graph).*** *A QoS-aware process graph consists in a process graph that is annotated with the expected load for each operation, along with the NFRs associated with the related service composition.*

Figure 2 shows two choreographies specified using this notation. At this stage, services are specified in an abstract way. They will be subsequently replaced by concrete implementations as a result of service selection.

Our proposal for choreography enactment is based on the combined representation of multiple choreographies using a structure called *QoS-Aware Dependency Graph.* This structure represents the services that are part of the choreographies, the dependencies among those services, and their NFRs.

Definition 6 ***(QoS-Aware Dependency Graph).*** *A QoS-aware dependency graph* $\mathcal{G}$ *is a directed graph represented by a tuple* $(\mathbb{P}, \mathbb{E}, \mathcal{Q})$*:*

- $\mathbb{P} = \{b \cup z \cup \mathcal{S}\}$ *is a set of vertices, where b and z represent the initial and end vertices, respectively.*
- $\mathbb{E}$ *is the set of directed edges. Each edge* $e \in \mathbb{E}$ *is a tuple* (p_s, p_r, o)*, where* $p_s \in \{\mathbb{P} - z\}$ *is the send vertex,* $p_r \in \{\mathbb{P} - b\}$ *is the receive vertex, and o is the operation being requested.*
- $\mathcal{Q}$ *is a set of QoS properties. Each* $q \in \mathcal{Q}$ *is a tuple* $(k, \Omega, \lambda, \phi)$*, where k is a QoS metric,* λ *is the load* ($\lambda > 0$)*,* $\phi \in \mathbb{D}_k$ *is the target average value for this metric, and* Ω *is a set of pairs* $(s \in \mathcal{S}, o)$ *that represent the services and the target operations to which the metric must be applied.*

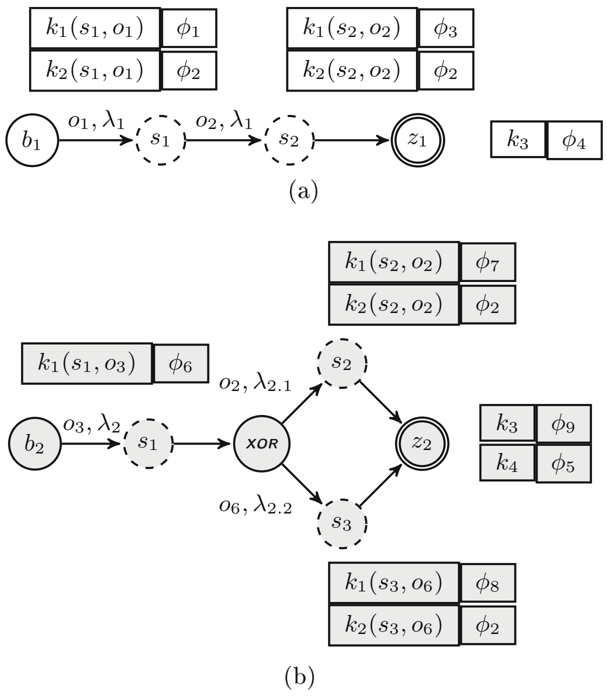
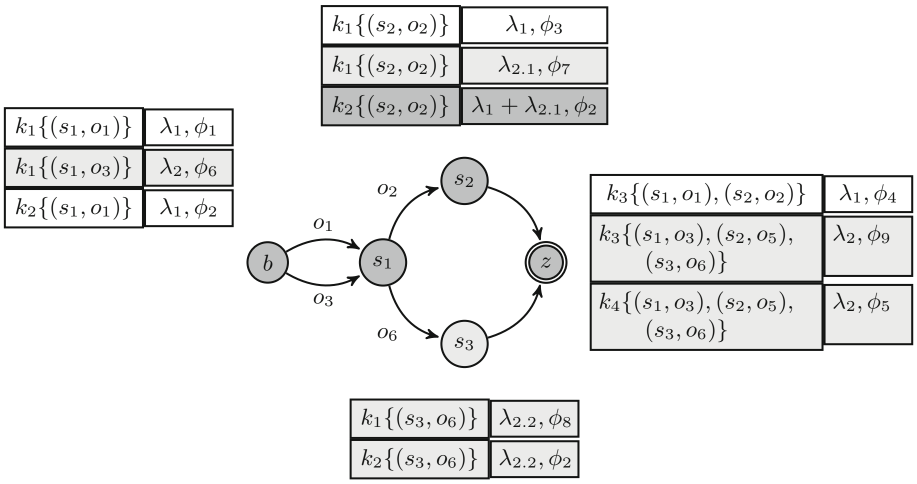


Fig. 2. Two choreographies specified using the QoS-Aware process graph notation.



$k_1\{(s_2, o_2)\}$ λ_1, ϕ_3
$k_1\{(s_2, o_2)\}$ $\lambda_{2.1}, \phi_7$
$k_2\{(s_2, o_2)\}$ $\lambda_1 + \lambda_{2.1}, \phi_2$
$k_1\{(s_1, o_1)\}$ λ_1, ϕ_1
$k_1\{(s_1, o_3)\}$ λ_2, ϕ_6
$k_2\{(s_1, o_1)\}$ λ_1, ϕ_2
b
o_1
o_3
s_1
o_2
s_2
o_6
s_3
z
$k_3\{(s_1, o_1), (s_2, o_2)\}$ λ_1, ϕ_4
$k_3\{(s_1, o_3), (s_2, o_5), (s_3, o_6)\}$ λ_2, ϕ_9
$k_4\{(s_1, o_3), (s_2, o_5), (s_3, o_6)\}$ λ_2, ϕ_5
$k_1\{(s_3, o_6)\}$ $\lambda_{2.2}, \phi_8$
$k_2\{(s_3, o_6)\}$ $\lambda_{2.2}, \phi_2$

Fig. 3. QoS-aware dependency graph for the choreographies shown in Fig. 2.

Figure 3 illustrates the QoS-aware dependency graph for the two annotated choreographies shown in Fig. 2. We can find elements that remain the same as in the original choreographies (shown in lighter shades of gray) as well as elements that had some change in their load and target values (highlighted in darker tones). Changes are due to the increased load on services and to the aggregation of NFRs when they have the same target. In this structure the services represent concrete chosen implementations.

This formalization enables the representation of combined choreographies and the execution of more realistic service selection and resource allocation. Additionally, these aspects must be reexamined (i.e., adapted) during choreography enactment (at runtime). In the next section we outline the approach we are developing to do this using the model described here.

6 Adaptive Approach to Choreography Deployment

The preceding sections discuss the issue of managing multiple choreographies at the same time in the presence of service sharing. Users in charge of choreography management must take into account the different roles of services and select the resources needed to run each service. This must be done at deployment time, and needs to be constantly reviewed at runtime to match QoS requirements.

The formalization presented in the previous section can be used to deal with the service sharing issue during choreography enactment. It facilitates the initial resource allocation and its adaptation at runtime as outlined next.

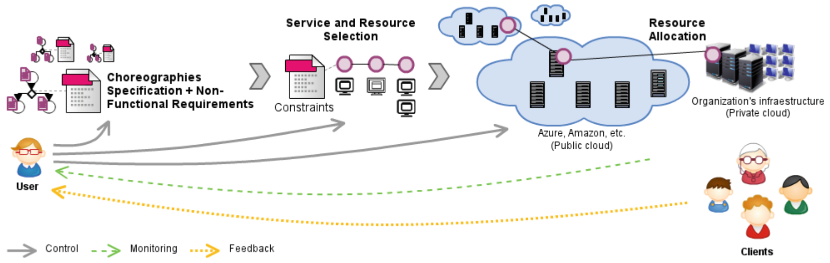

Choreographies Specification + Non-Functional Requirements
Service and Resource Selection
Constraints
Resource Allocation
Azure, Amazon, etc. (Public cloud)
Organization's infraestructure (Private cloud)
User
Clients
Control
Monitoring
Feedback

Fig. 4. Scenario of manual choreography enactment management.

As illustrated in Fig. 4, according to feedback from clients, such as regarding the level of satisfaction, or from the system, e.g., number of aborted requests, the user must manage service and resource allocation and adaptation. Every time some QoS violation is detected, the first attempt to deal with it is through adaptation of resource allocation. In cases where it is not possible to achieve the needed QoS by acting (solely) at this level, another strategy is to perform adaptation on service selection and resource configuration. As a last attempt, the user may be required to adapt the choreography and/or accept lower QoS.

Our approach to automate the above scenario is to use models at runtime [14]. The use of models at runtime allows the specification of services and resource requirements based on the current needs of applications; it also allows more precise management of the available computing power, especially compared to the allocation of resources based simply on profiles of virtual machines (VM). In doing so, service selection and resource allocation can be performed automatically according to abstract models and monitored data, thus facilitating adaptation.

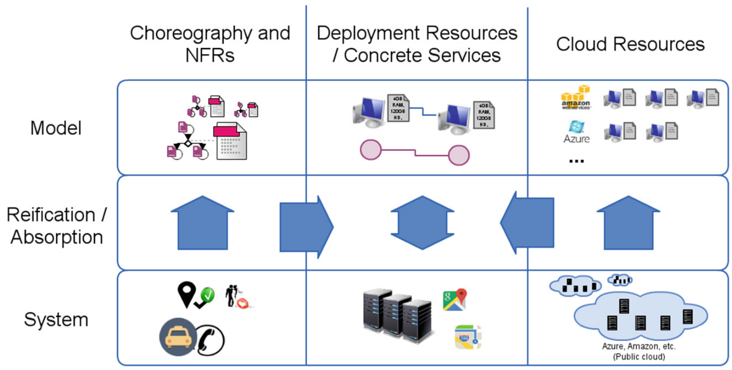


Fig. 5. Runtime models.

As can be seen in Fig. 5, our proposal relies on three different entities that are abstracted using models. The choreography model (upper left side in the figure) is represented using the QoS-aware process graph notation and is the input in our approach. It is then used to generate the deployment model (upper middle part in the figure), which is represented using the QoS-aware dependency graph notation (with concrete service selection). The dependency graph, in turn, is used to select the resources used to deploy/run the services. Moreover, the cloud resources model (upper right side) represents available resource configurations and is used as input for resource selection. The formalization proposed in this paper can be used to specify the first two levels of modeling. We aim to extend this formalization to represent cloud resources as well.

Although there is reification[1] of the running system in all models, direct absorption[4] only applies to the deployment model, since changes on it are directly reflected on the running system. Nevertheless, changes in the other two models are also reflected in a indirect way since they are used as input for service and resource selection. Note that this is ongoing work and an implementation of the

[1] Reification is the action of exposing the representation of a system in terms of programming entities that can be manipulated at runtime. The opposite process, absorption, consists in effecting the changes made to these entities into the system [15].

proposed approach is currently being developed. We are currently implementing the generation of dependency graphs by means of the combination of the target process graphs. Service and resource selection in turn are being implemented using a variation of the multiple-choice multi-dimension knapsack problem [16].

7 Final Remarks

The sharing of services among multiple service compositions has a significant effect on the overall provided QoS. Based on this observation, we advocate that performing choreography enactment without taking this into account is not a realistic approach. We present some experiments that demonstrate the problem and propose a formal model to represent QoS-aware service compositions.

We aim to use the formal model presented here to represent abstract service compositions. Taking these abstract compositions as input, we can automatically select the best services in order of satisfy associated non-functional requirements. Another important ongoing work is to extend the formalization presented here with a cloud resources model in order to provide a basis to implement the allocation of resources to run the selected services in a multi-cloud environment.

Acknowledgments. This work is supported by the Brazilian foundations FAPEG (calls # 04/2011, 12/2012 and 03/2013) and CNPq (grants # 249809/2013-3 and 473939/2012-6).

References

1. Strunk, A.: QoS-aware service composition: a survey. In: 2010 IEEE 8th European Conference on Web Services (ECOWS), pp. 67–74. IEEE (2010)
2. Barker, A., Walton, C.D., Robertson, D.: Choreographing web services. IEEE Tran. Serv. Comput. **2**(2), 152–166 (2009)
3. Zeng, L., Benatallah, B., Ngu, A.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for web services composition. IEEE Trans. Software Eng. **30**(5), 311–327 (2004)
4. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: QoS-aware replanning of composite web services. In: Proceedings of IEEE International Conference on Web Services, ICWS 2005, Proceedings, pp. 121–129. IEEE (2005)
5. Peng, X., Changsong, L.: ESCA: evolution-strategy based service composition algorithm for multiple QoS constrained cloud applications. Int. J. Future Gener. Commun. Netw. **7**(1), 249–260 (2014)
6. Field, T.: JINQS: an extensible library for simulating multiclass queueing networks, v1.0 user guide (2006). http://www.doc.ic.ac.uk/ajf/Software/manual.pdf. Accessed 30 March 2015
7. Nguyen, X.T., Kowalczyk, R., Han, J.: Using dynamic asynchronous aggregate search for quality guarantees of multiple web services compositions. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 129–140. Springer, Heidelberg (2006)
8. Ardagna, D., Mirandola, R.: Per-flow optimal service selection for web services based processes. J. Syst. Softw. **83**(8), 1512–1523 (2010)

9. Furtado, T., Francesquini, E., Lago, N., Kon, F.: A middleware for reflective web service choreographies on the cloud. In: Proceedings of the 13th Workshop on Adaptive and Reflective Middleware, vol. 9. ACM (2014)
10. Huang, K.C., Shen, B.J.: Service deployment strategies for efficient execution of composite SaaS applications on cloud platform. J. Syst. Softw. **107**, 127–141 (2015)
11. OMG: Documents Associated with Business Process Model and Notation (BPMN), Version 2.0 (2011). http://www.omg.org/spec/BPMN/2.0/
12. Mendling, J., Lassen, K.B., Zdun, U., et al.: Transformation strategies between block-oriented and graph-oriented process modelling languages. In: Multikonferenz Wirtschaftsinformatik, vol. 2, unknown, pp. 297–312 (2006)
13. Rosario, S., Benveniste, A., Jard, C.: Flexible probabilistic QoS management of transaction based web services orchestrations. In: IEEE International Conference on Web Services, ICWS 2009, pp. 107–114. IEEE (2009)
14. Blair, G., Bencomo, N., France, R.B.: Models@run.time. Computer **42**(10), 22–27 (2009)
15. Kon, F., Costa, F., Blair, G., Campbell, R.H.: The case for reflective middleware. Commun. ACM **45**(6), 33–38 (2002)
16. Khan, S., Li, K.F., Manning, E.G., Akbar, M.M.: Solving the knapsack problem for adaptive multimedia systems. Stud. Inform. Univ. **2**(1), 157–178 (2002)