USING FINITE STATE MODELS FOR QUALITY EVALUATION AT WEB SERVICE DEVELOPMENT STEPS

Olga Kondratyeva1,2, Natalia Kushik1,2, Ana Cavalli1, Nina Yevtushenko2
1TELECOM SudParis, Evry, France; 2Tomsk State University, Tomsk, Russia
olga.kondratyeva@telecom-sudparis.eu, ngkushik@gmail.com, ana.cavalli@telecom-sudparis.eu, ninayevtushenko@yahoo.com

Abstract
This paper presents a short study on evaluating the web service quality based on finite state models using the well-known metrics of Quality of Service (QoS) and Quality of Experience (QoE). These metrics represent objective and subjective assessments. The development of any web service involves several steps and the final values of the service QoS and QoE can be essentially improved/deteriorated based on the quality reached at each development step. In this paper, five service development steps are considered: those are requirements specification, provisioning, composition, implementation, and usage and management. At each step different finite state models for refining the service quality estimation are utilized. A running example of a vacation planner service illustrates the application of finite state models to improve the accuracy of the QoS and QoE evaluation.

Keywords: web service; quality of service (QoS); quality of experience (QoE); finite state models; attribute/parameter; composite service quality.

1. INTRODUCTION

Nowadays web services are used almost everywhere (Al-Masri & Mahmoud, 2008). Moreover, the number of services increases very fast in order to allow users an easy manipulation of various online applications. Different services developed for the same purpose can be found in service repositories (see, for example, Curbera et al., 2002). Nevertheless, the same functional properties do not mean that those web services have the same quality. Thus, in order to efficiently select the best web service among the great number of available services it is necessary to have the adequate evaluation of the service quality.

In most papers, the quality of a given web service is defined as a set or a pattern of attributes/parameters of this service (Khirman & Henriksen, 2002; Al-Masri & Mahmoud, 2008; Hyun-Jong et al., 2008; Morais & Cavalli, 2012). As it is mentioned in (Hyun-Jong et al., 2008), the major attributes to define the Quality of Service (QoS) are the time delay, the package loss percentage, the service access facility (the availability), the reliability, etc. All these parameters are rather objective and thus, can be evaluated automatically when a set of possible values is specified in advance for each parameter. Nevertheless, there are other attributes that also affect the user satisfaction with a web service, such as service design, how easy the service is to use, etc. Those parameters are subjective and their values significantly depend on human preferences. These informal service requirements together with QoS parameters are usually evaluated as the Quality of Experience (QoE) for the web service. In other words, when measuring or predicting the user satisfaction or User Experience (Winckler, Bach, & Bernhaupt, 2013) with respect to a web service both QoS and QoE should be taken into account.

In this paper, we focus on formal models for the web service quality evaluation, since formal models are now widely used to evaluate the quality of different software products. Web services can often be described by sets of permissible sequences of actions and finite state models have been proven to be well adapted to verify functional properties of web applications. In this study, finite state models are shown to be very helpful when evaluating the quality of web services.

As any software product, a web service passes different steps while being developed. Thus, web service quality can be eventually improved/deteriorated at each development step. This is the reason why a precise quality evaluation needs to be performed at each service development step. In this paper, we consider the following development steps: service requirements specification, provisioning, composition, implementation, and usage and management.

At the first two steps, the specification and provisioning steps, a set of service requirements as well as a set of available resources needed for the service implementation are determined. The service semantics and external interfaces are given by a formal language description of a service (Booth et al., 2004). One of the basic standard languages is the web service description language (WSDL) and a number of service depositories publish WSDL service files and provide the automatic analysis of such files (see, for example, XMethods depository).
However, in most cases, web services are represented as a composition of other available services. At the service composition step a complex service is designed as a composition of simpler ones. Special languages for the composite service description have been developed. Workflows which support the execution logic of composite complex services can be described using the business process execution language (BPEL). A service implementation significantly depends on a composition derived at the service composition step. Correspondingly, the composition and implementation steps are considered in the same section.

Different techniques can be applied to estimate the QoS at the specification, provisioning, composition, and implementation service development steps. In this paper, we focus on using finite state models for this purpose and discuss how different models are used at each service development step. For instance, we discuss that finite automata describing service functionality can be augmented with weights to take into account the service provisioning. We further show that sometimes workflows, which are often used for representing a composite service, are insufficient for the precise evaluation of its quality. At the last step of web service development, the usage and management step, the service becomes available for users and is adapted to user preferences, if needed. Thus, at this step basically the QoE is evaluated. The QoE estimation problem mainly refers to a problem of modeling user perception that is nowadays one of the most well-known problems in the artificial intelligent area. Different finite state models might be of a big help when predicting the QoE value of a given web service. Based on the QoE prediction using the developed models, decisions could be made for which groups of users the service can be attractive and what profit can be obtained. Another issue can be concerned about why the service is not attractive for some groups of users and what should be modified in order to obtain a higher profit of this service usage.

This paper is an extension of (Kondratyeva, et al., 2013a; Kondratyeva, et al., 2013b) where finite state models have been considered for the QoS and QoE evaluation. The main contribution of this paper is the study how the web service quality estimation can be refined at different development steps using finite state models. Each service development step and corresponding quality estimation technique are illustrated by using an example of a composite service that guides the paper.

The rest of the paper is organized as follows. Section 2 contains the preliminaries. Section 3 is devoted to evaluating the QoS of web services using finite state models at the specification and provisioning steps. Section 4 contains a discussion on the QoS estimation during the composition and implementation development steps. Section 5 is devoted to the brief description of finite state models for the QoE evaluation that are used at the usage and management service development step. Section 6 concludes the paper.

2. Preliminaries

A web service can be defined as a composition of web applications where a server (client) in one application can be turned into a client (server) in another one. A similar definition is given, for example, in W3C technical report (Booth et al., 2004), where a web service is defined as “a software system designed to support interoperable machine-to-machine interaction over a network”.

The QoS (Quality of Service) can be defined as a set of attributes (or parameters), such as the response time, availability, reliability, etc., corresponding to a given web service and allowing to compare and rank services with similar functionalities (Khirman & Henriksen, 2002; Al-Masri & Mahmoud, 2008; Hyun-Jong et al., 2008; Morais & Cavalli, 2012). This set of attributes is often mapped into a single value, quality score, using an appropriate computable function and the result of this function can be an integer, a rational, a (fuzzy) logic constant, etc. The QoS parameters are rather objective and thus, can be evaluated automatically when a set of possible values is specified in advance for each parameter.

Despite of the fact that objective quality attributes are important for having a good service, they are insufficient to guarantee that a user is satisfied with the service, since user satisfaction may be essentially affected by some other attributes, such as service design, ease of use, etc. Those parameters together with the satisfaction itself are highly subjective and significantly depend on user preferences, psychological and physiological features of human perception, his/her mood and other hard predictable reasons. The subjective evaluation of the received service is usually represented with the Quality of Experience (QoE). For better quality-aware management of web services, researchers study the automatic QoE estimation and prediction based on the values of QoS attributes (Lalanne, Cavalli, & Maag, 2012).

Running example. In order to illustrate how finite state models can be used at different development steps for the quality evaluation, an example of a vacation planner service is considered throughout the paper. This example with slight modifications is taken from (Escobedo Del Cid, 2011; Gaston & Le Gall, 2012). The service allows a user to purchase flight tickets and to book an accommodation at the destination point. A user submits traveling dates and the planner proposes a number of available options for flight tickets and hotel rooms. If the user and planner agree on the flight ticket and hotel room then the vacation is successfully booked. Otherwise, the vacation reservation has failed.

Service development steps. As mentioned in the introduction, web services are developed in several major steps which can be defined in different ways starting from classical software development life cycle models (ISO/IEC,
Section 3.1 discusses how the quality requirements of the service under development can be selected. Section 3.2 focuses on the relationships between selected requirements as this is important when comparing the quality of two services with the similar functionalities. In Section 3.3, it is briefly discussed how finite state models extracted from the service functional description can be used for the quality evaluation. In Section 3.4, the extracted formal model is augmented with weights according to the available information about service provider and/or the experience of using services with similar functionalities. Such augmentation sometimes allows a more precise quality evaluation.

3.1 Specifying Web Service Requirements

The requirements for the service under construction can be implicitly divided into two groups: functional and non-functional requirements. In this paper, the requirements that affect the service quality are studied. Correspondingly, at the first step it is important to specify which quality parameters are crucial for the service quality and this choice essentially depends on web service features. The choice of quality parameters has a significant impact on the further attractiveness of the constructed service.

Example. For the vacation planner the following functional requirements can be set. 1) Vacation is booked if both, a flight ticket and a hotel room, are reserved. 2) Flight tickets are proposed before booking a hotel room.

The profit of the vacation planner essentially depends on how fast the requests can be processed and on the availability and popularity of the service. Correspondingly, the response time, service availability and service popularity are considered as crucial parameters in this example. In the running example, after marketing analysis it can be stated that 1) each user request has to be processed in at most 30 seconds, and 2) at least 20% of users who are interested in vacation planning should book the vacation using this service.

In the current section, the formalization of quality requirements is discussed. Functional requirements are formalized in the corresponding specification (Section 3.3).

The list of crucial parameters is usually defined as the quality vector $QoS = \langle q_1, q_2, ..., q_n \rangle$. The restrictions on parameter values (if there are any) can be, for instance, stated by a system of equations or inequalities. Equivalence and order relations can be used for specifying the priorities over quality parameters (Xin et al., 2013). For example, the notations below can be used:

- $q_i > q_j$ – parameter $q_i$ is more important than parameter $q_j$;
- $q_i \gg q_j$ – parameter $q_i$ is much more important than parameter $q_j$;
- $q_i \approx q_j$ – parameters $q_i$ and $q_j$ have the same importance.

The more precise is the service quality evaluation at each development step the higher is the probability that the developed service is rather efficient. In this paper, at each development step the use of finite state models for the more precise quality evaluation is analyzed. A running example illustrates how the use of these models can refine the service quality evaluation.

3. SERVICE REQUIREMENTS SPECIFICATION AND SERVICE PROVISIONING

This section focuses on formalizing functional and non-functional requirements for the service under construction.
3.2 Comparing the Quality of Two Services

To be able to produce a "better" service than those which already exist, it is necessary to define the notion "to be better" for two services. In this study, two approaches for such evaluation are briefly sketched: the use of utility function and Pareto-dominance relationship. Utility function (Sun & Zhao, 2012; Lalanne, Cavalli, & Maag, 2012; Cardoso et al., 2004) is a computable function that maps the QoS vector into a single quality score \( F(QoS) \). In other words, given two services \( S_1 \) and \( S_2 \) with the quality vectors \( QoS_1 \) and \( QoS_2 \), the service \( S_1 \) is better than \( S_2 \) if \( F(QoS_1) > F(QoS_2) \). There are many ways for defining utility functions; the simplest option is to define such a function as a weighted sum of quality parameter values, \( F(QoS) = \sum_{q_i \in QoS} w_i q_i \) where the main question is how to choose weights. In (Sun & Zhao, 2012), weights are chosen in such a way that a weighted parameter value is between 0 and 1 where 0 corresponds to the "worst" parameter value while '1' corresponds to the "best". In (Xin et al., 2013), the weights are calculated based on a partial order relation between parameters. The "worst" and "best" parameter values essentially depend on a parameter. For instance, for the response time the less the value is, the better it is for the service quality, while for the availability or popularity a maximal discrepancy value can be set for these parameters.

Example. For the vacation planner the quality vector is \( QoS = (t, a, r) \), where \( t \) denotes the response time, \( a \) is the availability and \( r \) is the popularity. According to the quality requirements and marketing analysis, those parameters can be formalized in the following way:

\[
\begin{align*}
    t &\leq 30 \\
    r &\geq 0.2
\end{align*}
\]

The system (1) establishes the limitations on the values of the response time and popularity, while the system (2) establishes a partial order relation on the set of parameters and states that the response time has the priority with respect to the availability and popularity.

3.3 Deriving a Formal Specification

Web services published in repositories are often described using XML-based languages, such as WSDL, BPEL, WS-CDL (Web Service Choreography Description Language) depending on abstraction and hierarchical levels, etc. (Booth et al., 2004). As web services often process permissible sequences of actions, finite state models are widely used for their analysis and synthesis. In (Rabanal, Rodriguez, Mateo, & Diaz, 2012), the authors extract a Finite State Machine (FSM) from the WS-CDL description using the tool DIEGO 2.0. In (Diaz et al., 2005), the BPEL and WS-CDL descriptions are translated into a system of communicating timed automata. In most papers, the extracted finite state models are used for automatic composition of services (Beek, Bucchiarone, & Gnesi, 2006) or testing and verification purposes (Tien-Dung, Felix, & Castanet, 2010; Honghao & Ying, 2011). As mentioned in (Héam, Kouchmarenko, & Voinot, 2010; Kondratyeva et al., 2013), finite state models can also be helpful for more precise quality evaluation.

Example. For the running example, the service finite automaton specification at a high abstraction level is given in Figure 1.

Figure 1. An automaton for the vacation planner web service

At the initial state \( q_0 \), the vacation planner "asks" a user to define preferable dates of traveling and replies at state \( q_1 \) with the dates available for plane tickets. The user declares his/her opinion (agree/disagree) about the ticket and the service moves to state \( q_3 \) where two options are available. If the travel conditions do not satisfy the user then the system moves to state \( q_2 \) where the user can change the travel dates and continue or can quit the service via state \( q_4 \). If the user accepts the ticket conditions provided by the service at state \( q_3 \) then the service proposes the dates for booking a hotel room, moves to state \( q_5 \) and "asks" the user if he accepts the proposed conditions for the accommodation. If the conditions are accepted (not accepted) the output is 'reserved'

http://hipore.com/ijsc
(not reserved). In both cases, the system moves to the final state (the same as the initial one).

An extracted formal model can be hierarchical. For example, if more detailed and precise description is needed, it is possible to unroll a composite state $q_3$ as shown in Figure 2.

![Figure 2. An automaton for representing a composite state $q_3$ of the vacation planner](image)

When considering a state $q_3$ as a composite state the input ?fstat is unrolled into a sequence of four inputs, namely, into inputs ?class, ?travel time, ?departure time and ?price. The input ?class allows a user to specify the class of a requested flight, i.e., a business or an economy class, while the input ?travel time refers to a maximal number of hours required for traveling. The input ?departure time specifies the earliest time for a tourist to depart while the input ?price is related to the maximal price of a plane ticket. Inputs ?reserve and ?not reserve correspond to the flight booking or canceling the order. The automaton outputs belong to the set {?class, ?travel time, !departure time, ?price, [reserved]} and each input is followed by the same output. For instance, if a user specifies a preferable class by applying the ?class input the service outputs possible class options with the output ?class. If the class options do not satisfy the user he/she can apply the ?not reserve input and the system moves to state $q_3$ where the output !not reserved is produced. In other words, at state $q_3$ two outputs can be produced, i.e., the output ?date that corresponds to !reserved output at state $q_3^{10}$ or the output !not reserved that corresponds to the same output produced at the state $q_3^{12}$.

### 3.4 The Service Provisioning Step

The service provisioning step includes the analysis of available and required resources for the service under development. This issue involves business processes and related resources such as human-hours to be spent, the cost of implementation of a new service comparing to the usage of existing components, etc. All these parameters significantly influence the system architecture as well as a choice of component services and the implementation quality but their influence is rather implicit and for this reason, such business issues are left out of the scope of this paper. Nevertheless, knowing some network resources (server quality, internet speed, etc.) gives a chance for more precise evaluation of some service parameters such as the response time or availability of the service. The marketing analysis of people who are going to use such service can also refine the evaluation. The quality evaluation at this step can become a part of Service Level Agreement (SLA) (Keller & Ludwig, 2002). If the predicted quality is under desirable standards then the service requirements specification should be revised, i.e., a developer should come back to the previous development step.

Given a finite state model extracted from some service description, at the provisioning step the model can be refined by augmenting its states and/or transitions with weights, probabilities or other attributes based on the quality requirements.

In particular, in (Héam, Kouchnarenko, & Voinot, 2010), weighted automata are used for more precise service quality evaluation. The weight associated with each automaton transition represents the cost of the corresponding transaction execution (in terms of time, money, etc). The quality parameter values can be estimated via different execution paths of the corresponding automaton and some conclusions about the service quality can be drawn.

**Example.** In order to illustrate how the QoS of the vacation planner service can be evaluated at the provisioning step, the automaton in Figure 1 is augmented with weights for transitions that are responsible for service transactions. In this example, the weight is interpreted as a time needed for a corresponding service transaction, and a corresponding weighted automaton is shown in Figure 3. It is assumed to take at most 10 seconds for the vacation planner to check and to provide the available travel dates and not more than 3 seconds to confirm whether a ticket or a hotel room is reserved. The reservation of a ticket or a hotel room requires 10 seconds and thus, checking dates for a hotel room after booking the ticket lasts at most 13 seconds.

![Figure 3. A weighted automaton for the vacation planner service](image)
specification can be considered. The result is 10 + 3 + 1 + 13 + 3 + 6 = 36 seconds. When using the formal model of a weighted automaton (Figure 3) the QoS can be evaluated more accurately by considering traces from the initial to the final state. In this case, as mentioned in (Heam, Kouchnarenko, & Voinot, 2010), the weight of each simple trace (without cycles) may be calculated and the longest transaction time corresponds to a trace with the maximal weight. In the example, this trace labels the sequence of states $q_0 q_1 q_2 q_3 q_5 q_6 q_0$ with total weight 29. Thus, the refined maximal transaction time for the vacation planner equals 29 seconds.

4. SERVICE COMPOSITION AND IMPLEMENTATION

In this section, we briefly discuss the main aspects of deriving a quality-aware composite service. When a new service is derived by composing already existing web services, possibly developed by a third-party, two main questions arise: what service quality can be provided by existing components, and how to select components to achieve a higher quality of composite service. These two questions are considered in Section 4.2 and Section 4.3 correspondingly, while Section 4.1 presents the approach for a composite service architecture specification.

4.1 Basic Composition Patterns

For quality evaluation purposes, the functioning of the component services and the content of the messages they exchange are not considered. When designing the architecture of a composite service, task invocations between components can be expressed with a corresponding workflow and Figure 4 illustrates the basic composition patterns in which the workflow can be decomposed. When a component service is invoked, it executes some task (according to the composition requirements), and after completing the task, the service either produces the result if it is the final task, or invokes other components to execute further composition tasks. The same component service can be used for executing different composition tasks. To avoid any ambiguity further the execution of some task by the service is referred to as a component service.

The simplest workflow compositional pattern is sequential (Figure 4a) where the composite service is organized as follows: when the service $S_0$ completes a task then the service $S_1$ is invoked. In a conditional pattern (Figure 4b), also referred to as XOR-split pattern, $S_0$ invokes one and only one of services $S_1, ..., S_k$ depending on the results of the task execution. When probabilities are involved for each possible invocation, the equality $\sum_{i=0}^{k} p_i = 1$ must be held. When the service $S_0$ invokes several services $S_1, ..., S_k$, a parallel pattern (Figure 4c), or an AND-split pattern, is considered. Services executing tasks in parallel can be further merged in order to execute a required later task. If the next service is invoked only when all preceding services have completed their tasks, the synchronizing pattern (Figure 4d), or AND-joint pattern, is considered. Otherwise, if the next service is invoked after at least one service completes its task, a concurrent pattern (Figure 4e), or a XOR-joint pattern, is at hand. When some tasks should be repeated, the loop pattern (Figure 4f) is involved, and the number of repetitions may be either known a priori or can be calculated during the task execution. Without loss of generality, loops are considered to have a single starting point (service $S_1$ in Figure 4f) and any number of exits (service $S_k$ in Figure 4f). For each exit point, the probabilities of continuing the loop or of going out may be specified.

![Figure 4. Basic compositional patterns: (a) sequential, (b) conditional, (c) parallel, (d) synchronizing, (e) concurrent, (f) loop](http://hipore.com/ijsc)

**Example.** The vacation planner service can be represented as the composition of two services: Flight Booking (FB) and Hotel Booking (HB) services. In the workflow in Figure 5, nodes $S$ and $F$ are initial and final nodes correspondingly.

![Figure 5. Workflow for the vacation planner service](http://hipore.com/ijsc)

After asking for a flight ticket a user has three options: to ask for a hotel room when flight dates are set; to change flight dates or to quit the service when the flight dates cannot be changed for some reason. In the case of booking a hotel room, a request is processed only once. In both cases when a hotel room is booked or not booked, the service is quit.
4.2 Aggregation Functions

Given a set \( \{S_1, S_2, \ldots, S_i\} \) of component services, their QoS vectors \( \{Q_1, Q_2, \ldots, Q_i\} \) (for instance, published by service providers or so-called service brokers (Zheng et al., 2013)), and the composition structure, the question is: what is the quality \( Q \) of the composite service?

For all basic patterns and their combinations the question of overall QoS evaluation has been studied properly and aggregation functions have been elaborated for a number of QoS parameters. The result of each aggregation function is the value of a corresponding attribute of the composite service. Some of these functions without considering probabilities for XOR-splits are summarized in Table 1. In Table 2, probabilities for XOR-splits and loops are taken into account when deriving aggregation functions. In these tables, the integer \( k \) denotes the number of involved services while the integer \( n \) is used for the number of loop iterations, and \( p_i \) is the probability of invoking the service \( S_i \).

In fact, the set of functions in Table 1 is incomplete, since more attributes can be considered such as popularity, especially for social networks (Billionaire, Greiman, & Goshia, 2009). When involving some logic models, such as fuzzy logic, \( k \)-value logic etc., or modular arithmetic models, aggregation functions which use only sum and multiplication operators do not seem to be sufficient and thus, novel corresponding aggregation functions have to be elaborated.

Table 2 contains no aggregation functions for sequential and parallel patterns since they coincide with those in Table 1. For a conditional pattern the average quality evaluation is calculated, though the quality often is computed for each path separately, which allows to assess the worst case, the best case, and/or the quality along the most probable execution path (Zheng et al., 2013). When probabilities are given, the quality of a loop pattern is calculated for an arbitrary number of iterations.

\[ \sum_{i=1}^{k} c_i \]

\[ \prod_{i=1}^{k} a_i \]

\[ \sum_{i=1}^{k} t_i \]

\[ \prod_{i=1}^{k} b_i \]

\[ \sum_{i=1}^{k} \frac{1}{c_i} \]

\[ \prod_{i=1}^{k} (1 - a_i) \]

\[ \max(t_i) \]

\[ \min(t_i) \]

\[ \sum_{i=1}^{k} \frac{1}{b_i} \]

\[ (\prod_{i=1}^{k} \frac{1}{a_i})^n \]

\[ \sum_{i=1}^{k} \frac{1}{b_i} \]

\[ (\prod_{i=1}^{k} \frac{1}{b_i})^n \]


4.3 Quality-Aware Component Service Selection

When the estimated quality of a composite service is unsatisfactory the question arises how the quality could be enhanced. According to the previous section, the quality of a composite service significantly depends on the composition structure and the quality of component services. The composition structure is mainly pre-determined by functional requirements and hence, cannot be easily changed. Correspondingly, in order to enhance the quality of a composite service there is an option of selecting a “better” component service, a so-called the quality-aware component selection problem. A “better” component service can be selected from a collection of services with similar functionalities based on a corresponding utility function or a skyline order relation.

The quality-aware component selection is a multi-dimensional optimization problem and often can be reduced to the well-known combinatorial problems such as the multi-choice knapsack problem (e.g., Xiaopeng & Chunxiao, 2009), the resource constraint project scheduling (Jaeger, Rojec-Goldmann, & Muhl, 2004), or the derivation of a shortest graph path under some constraints (Tao & Kwei-Jay, 2004), etc. Local and/or global approaches can be applied (Sun & Zhao, 2012, Xiaopeng & Chunxiao, 2009; Wu & Hong, 2010). Local selection approaches focus on choosing the best component for each task independently of other tasks, while the global optimization objective is to derive a composition with a better overall quality. For partially specified or derived on-the-fly compositions a local selection approach is reasonable, though, it cannot guarantee that the composite service satisfies the given quality requirements (Zeng, et al. 2003).

http://hipore.com/ijsc
conclude that for the vacation planner when both flight ticket and hotel room are booked, we specify. Based on the definition of the service popularity, though, the aggregation function should be corresponding aggregation functions, the composite service composition are implemented. At this step, detecting the SLA violations becomes one of the main service management objectives. Functional violations are out of the scope of this paper; in this section, we only discuss how the service effectiveness can be evaluated during the step of usage and management. The data collected via monitoring of service activities fail to be used. If some parameter values do not satisfy a developer he/she can come back to the previous steps in order to redesign the service.

5. SERVICE USAGE AND MANAGEMENT

The main objectives of service management include, but are not restricted to, improving service quality and ensuring that the service satisfies functional and non-functional requirements (Booth et al., 2004; IBM, 2001). Requirements that service should meet can be specified in the Service Level Agreement (Keller & Ludwig, 2002) and are checked via service monitoring when various data are collected and analyzed. At this step, detecting the SLA violations becomes one of the main service management objectives. Functional violations are out of the scope of this paper; in this section, we only discuss how the service effectiveness can be evaluated during the step of usage and management. The data collected via monitoring of service activities fail to completely illustrate the service profitability, since the main purpose of each service is to satisfy an end-user. Still, there is no confidence that a user is satisfied even when the QoS is rather high. For this reason, at the management step the QoE value is involved. It should be noted that the QoE can be evaluated during the step of usage and management.

Global approaches which can ensure the composite service quality utilize the multi-dimensional optimization (Zeng, et al. 2003; Wu & Hong, 2010; Moser, Rosenberg, & Dustdar, 2012) and these approaches are rather computationally expensive. Novel promising approaches are rather the mixture of local and global approaches (Sun & Zhao, 2012; Xin, et al., 2013; Shaoqian, Wanchun, & Jinjun, 2013).

Example. For the running example, the QoS is the vector \( QoS = (t, r, a) \), and based on the workflow in Figure 5 and corresponding aggregation functions, the composite service response time and availability can be calculated. To calculate the popularity, though, the aggregation function should be specified. Based on the definition of the service popularity and regarding that the vacation is booked successfully only when both flight ticket and hotel room are booked, we conclude that for the vacation planner service the popularity could be calculated as the product of popularities of services FB and HB. Consider three different hotel booking services HB1, HB2, HB3 with the QoS vectors \( QoS1 = (18, 0.5, 0.9), \) \( QoS2 = (20, 0.7, 0.9) \), and \( QoS3 = (10, 0.8, 0.6) \), correspondingly. If the flight booking service FB has the QoS vector \( QoFS = (15, 0.8, 0.9) \), then for the corresponding composite services, it holds that \( QoS_{C1} = (63, 0.4, 0.65), \) \( QoS_{C2} = (65, 0.56, 0.65) \), and \( QoS_{C3} = (55, 0.64, 0.44) \). As stated above, the popularity and the availability are assumed to be less important than the response time. Since all solutions comply the restriction that the popularity is at least 0.2, the service HB1 can be chosen as a component service due to the best response time.

4.4 The Implementation Step

When all decisions are taken about the service hierarchy and components, the non-existing component services and the overall composition are implemented. At this step, formal descriptions using finite state models can essentially help as there are many tools which allow automatic code generation (see, for example, Mtsweni, 2012). A developed implementation is then verified and tested for checking that the implementation conforms to its specification (Tien-Dung, Felix, & Castanet, 2010; Lallali, et al., 2008; Honghao & Ying, 2011). Other properties of the implementation such as security and robustness can be also tested at this level. Nevertheless, the quality related features which are expected according to the service specification usually are tested at the next step when the service is started to be used. If some parameter values do not satisfy a developer he/she can come back to the previous steps in order to redesign the service.
the QoE is discussed, since a direct feedback from users is not always available.

5.1 Traffic Monitoring for Quality Evaluation

Based on traffic analysis some statistics can be collected, such as the maximal time delay between transactions, the probability of losing a package, bandwidth etc. (Khirman & Henriksen, 2002). It is possible to consider how these parameters influence the traffic QoS evaluation. However, the above parameters are more related to a transport level than to the service under investigation itself. The choice of parameters to be monitored during the service usage is important for further management and concluding how each parameter influences the end-user satisfaction and how the service can be modified in order to improve its quality.

The main problem for the QoE evaluation is that this metric is highly subjective. Therefore, the most common and straightforward way to evaluate the user satisfaction is to use experts or directly ask a user through a questionnaire. Questions in a questionnaire can widely vary depending on which key parameters a service provider is interested in: whether a user has a confidence in the service, or whether a user is going to use the service again and recommend it to his/her friend, etc. (Dale et al., 2007). The questionnaire is then analyzed and a formal model for the QoE computing/prediction can be developed. Based on this model decisions could be made for which groups of users the service can be attractive and which business profit can be obtained. Another issue can be concerned about why the service is not attractive for some groups of users and what should be modified in order to make it more attractive.

Example. For the running example, during the service usage two parameters are monitored: the overall response time and the service popularity. The QoE score is supposed to be evaluated through some appropriate feedback procedure. The response time is an integer and is measured in seconds while service popularity is a rational between 0 and 1. The QoE score is an integer between 1 (the worst score) and 5 (the best score). The corresponding sample is presented by Table 3.

<table>
<thead>
<tr>
<th>Response time (sec)</th>
<th>Popularity</th>
<th>QoE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.5</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>0.8</td>
<td>4</td>
</tr>
<tr>
<td>30</td>
<td>0.4</td>
<td>2</td>
</tr>
</tbody>
</table>

Example. In the running example, during the service usage two parameters are monitored: the overall response time and the service popularity. The QoE score is supposed to be evaluated through some appropriate feedback procedure. The response time is an integer and is measured in seconds while service popularity is a rational between 0 and 1. The QoE score is an integer between 1 (the worst score) and 5 (the best score). The corresponding sample is presented by Table 3.

5.2 State Model Based QoE Evaluation

Having a statistical correspondence between QoS values and QoE scores after a number of experiments, one may derive complex approximation functions for automatic calculation of QoE based on the known QoS value (see, for example, Lalanne, Cavalli, & Maag, 2012). Analyzing those approximation functions a provider can estimate which parameters are crucial for the QoE evaluation. Unfortunately, it is not always possible to derive approximation functions, and thus, more complex analysis is required for automated QoE prediction.

Through the question how the human brain works still remains unsolved, advanced studies of cognitive processes can help to elaborate adequate models of human perception and decision making procedures. Human behavior being nondeterministic or even random can hardly be reduced to some formal algorithms. Therefore, self-adaptive models that can be trained are promising for user experience evaluation. Some of self-adapting models can be considered as finite state models and almost all of them rely on pre/post conditions that can be expressed in terms of IF-THEN operator such as “if the service works fast I am rather happy with this service” (Kondratyeva et al., 2013b). Artificial intelligence techniques, such as fuzzy logic and neural networks seem to be promising for establishing correspondence between measured objective quality parameters and user satisfaction grades. For example, in (Pernici & Siadat, 2011; Pokhrel et al., 2013) fuzzy logic is used to calculate the correlation between network parameters and the QoE value. Since this correlation differs from service to service and is usually significantly non-linear, it is essential to introduce adaptive algorithms to derive function coefficients automatically for given services. In (Pernici & Siadat, 2011) neural network learning algorithms are used to develop the fuzzy logic based tool to determine values and grades (excellent, good, average, or poor) for services. Talking about finite state models, a decision tree can be efficiently used for the QoE evaluation/prediction, as it is illustrated for the running example. Each node of a decision tree represents corresponding service parameter that is evaluated at a tree level, each transition corresponds to the possible parameter value.

Example. In the running example, corresponding decision tree has three levels and the root is labeled by the most important quality parameter, i.e., the response time (t). The second level of the tree contains nodes that are related to the service popularity (r). Terminal nodes of the tree are labeled by corresponding QoE values (Figure 6).

Figure 6. The decision tree for the vacation planner service

When observing the t value one can conclude about possible QoE values at terminal nodes in a corresponding subtree of the decision tree. For example, when the response time is 10 seconds, and the service popularity is 0.5, the QoE score is 3. When the response time is 30 seconds and the service popularity is 0.4, the QoE score is 2.
time equals 10 possible subtree branches lead to terminal nodes with the QoE that is at least 3. When new statistical data are gathered the decision tree should be adapted to these data by adding new branches. When predicting the QoE for parameter values which do not label a path from the root to a terminal node, additional rules should be set. For example, a branch with minimal discrepancy in parameter values can be selected for the QoE estimation. A decision tree can also be used for the analysis of the parameter importance for the users’ satisfaction based on the difference in the QoE score between neighbor subtrees. Readers may refer to (Kondratyeva et al., 2013b) in order to find out more about other finite state models that are often used for the automatic prediction of the QoE value.

6. CONCLUSIONS
In this paper, the advantages of using finite state models for the web service automatic QoE and QoS evaluation have been discussed. At each service development step, an appropriate finite state model has been considered for refining the quality evaluation. The running example of the vacation planner service clearly illustrates that finite state models relying on input/output sequences of service specification improve the accuracy of the QoS assessment. Since the user satisfaction is a priority of each web service, finite state models for the QoE evaluation that can be derived at the usage and management step have been briefly discussed. However, the development of new adequate and scalable formal models for the automatic QoE evaluation remains a challenging problem. Moreover, other neoteric quality metrics such as Quality of Design or Quality of Business are left out of the scope of this paper and need additional research.

7. ACKNOWLEDGMENT
The work is partly supported by ANR PIMI project (France). Olga Kondratyeva gratefully acknowledges the support of Futur et Ruptures scholarship from Fondation Telecom (France) and a scholarship of the President of Russian Federation for gifted students.

8. REFERENCES
Cubera, F., Duflot, M., Khalaf, R., Nagy, W., Mukhi, N., & Weerawarana, S. (2002). Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. IEEE Internet Computing, 6(2), 86-93, doi: 10.1109/4236.991449
International Journal of Services Computing (ISSN 2330-4472) Vol. 1, No. 1, October-December 2013

the 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC2004), 149-159, doi: 10.1109/EDOC.2004.1342512


Web service depository XM3ethods. from http://www.xmethods.net


Authors

Olga Kondratyeva has received her Master degree in Radiophysics (profile: Information systems and technologies) from Tomsk State University, Russia, in 2012. Currently she is a PhD student at double-degree program between Telecom SudParis, France and Tomsk State University, Russia. Her research interests include automata theory, service quality evaluation, timed-constrained systems, communicating systems.

Natalia Kushik has received her diploma degree in Applied Mathematics from Tomsk State University, Russia, in 2010. She has got a PhD degree in 2013 and worked as a PostDoc at Telecom SudParis, France. Her research interests include automata theory,
service quality evaluation, software testing and verification.

Professor Dr. Ana Rosa Cavalli is professor at Telecom SudParis and the head of Software and Networks department since 1990. Her research interests are on specification and verification, testing methodologies for conformance and interoperability testing, active testing and monitoring techniques, the validation of security properties and their application to services and protocols. She has had more than 200 peer review papers published in international conferences and journals. She led and participated recently in numerous national (CARRIOCAS, EXOTICUS, POLITESS, E-COMPAGNON) and European projects (ASK IT, E-LANE, SHIELD, IPNQSS, DIAMONDS, INTER-TRUST) contributing to the modelling, monitoring, testing, QoS and QoE analysis. She is member of several Steering and Program Committees of numerous international conferences: IEEE ICNP, IEEE ICST, IFIP CFIP, SARSII, AMOST-T, SETOP, IEEE ICSTSS, QESS. She has been co-chair of TESTCOM’95, SDL’97, PSTV/FORTE’98, ICNP’2002 et CFIP’2003. She has been domain chair of ICNP 2009, co-chair of the IEEE ICST 2010 and SETOP 2010 conferences.

Professor Nina Yevtushenko received her diploma degree in Electrical Engineering from Tomsk State University. Until 1991 she worked as a researcher with the Siberian Scientific Institute of Physics and Technology. From 1991 she joined Tomsk State University as a professor and presently she leads a research team working on the synthesis and analysis of discrete event systems. She also stayed as a visiting researcher/professor at Moscow State University, Université de Montréal, University of Ottawa, TelecomSudParis (former: Institute National des Telecommunications), Verona University. She published four books and many research papers. Her research interests include formal methods, automata theory, distributed systems, protocol and software testing.
SERVICES COMPUTING