QoS-aware dynamic service composition in ambient intelligence environments

Sonia Ben Mokhtar, Jinshan Liu, Nikolaos Georgantas, Valérie Issarny

QoS-aware Dynamic Service Composition in Ambient Intelligence Environments*

Sonia Ben Mokhtar, Jinshan Liu, Nikolaos Georgantas and Valérie Issarny
INRIA Rocquencourt, BP 105
78153 Le Chesnay Cedex, France
{Sonia.Ben_Mokhtar,Jinshan.Liu,Nikolaos.Georgantas,Valerie.Issarny}@inria.fr

ABSTRACT

Due to the large success of wireless networks and handheld devices, the ambient intelligence (AmI) paradigm is becoming a reality. One of the most challenging objectives to achieve in AmI environments is to enable a user to perform a task by composing on the fly networked services available at a specific time and place. Towards this goal, we propose a solution based on semantic Web services, and we show how service capabilities described as conversations can be integrated to perform a user task that is also described as a conversation, further meeting the QoS requirements of the user task. Experimental results show that the runtime overhead of our algorithm is reasonable, and further, that QoS-awareness improves its performance.

Categories and Subject Descriptors: D.2.12 [Software Engineering]: Interoperability

General Terms: Algorithms, Experimentation.

Keywords: Semantic Web services, OWL-S, Web services composition, QoS-awareness, Automata theory.

1. INTRODUCTION

Ambient Intelligence (AmI) envisions human-centric retrieval and consumption of information, in contrast to the conventional computer-centric approach. Systemically, this is realized as a synergistic combination of intelligent human-machine interfaces and ubiquitous computing and networking. One of the most challenging issues in AmI environments is to compose user applications by integrating on the fly networked software components that are available at a specific time and place, without any previous knowledge of these components. This would allow a mobile user carrying a handheld device with limited resources, to execute potentially complex applications in different AmI environments, benefiting from the richness of each such environment. Building upon semantic Web services (OWL-S), we introduce an approach to the QoS-aware dynamic composition of *user tasks* from networked services in AmI environments, which are user applications described in the form of a workflow (conversation) realizing a user's goal. This workflow description resides on the user's handheld device and is *abstract*, that is, it does not refer to any specific service to be integrated. Furthermore, the task description specifies the QoS required by the user. Accordingly, networked services have a description specifying their supported conversation and QoS. We propose a composition algorithm that integrates the networked services' conversations into a task's workflow, further considering QoS. Moreover, our solution introduces an abstraction of OWL-S conversations as finite state automata, and a QoS model.

The remainder of this paper is structured as follows. First, we present our approach of modeling and measuring QoS (Section 2). We then introduce our approach of modeling abstract user tasks and networked services (Section 3) and to the dynamic composition of tasks (Section 4). Further, we describe our prototype implementation and evaluation (Section 5). Finally, we conclude with a summary of our contribution and future work (Section 6).

2. MODELING AND MEASURING QOS

We introduce a base QoS specification of services that is suitable to AmI environments. Usually, a small number of parameters is sufficient to capture the dominant QoS properties of a system [3]. Along with the factor of limited resources on mobile devices, we only take into account the most dominant and descriptive dimensions in our base QoS specification, instead of trying to incorporate every possible applicable dimension. However, it can be easily extended with more dimensions, if required by specific services, by supporting the new dimensions in a way similar to the one discussed in this section. Below is the table of dimensions we introduce for specifying non-functional properties of services in AmI environments.

The values of quantitative QoS dimensions can be provided by QoS measurements using available software utilities (e.g., pathchar[1] for bandwidth measurement). However, providing values of metrics in service advertisements requires QoS prediction. QoS prediction prior to actual service execution can be carried out based on histories [4], which has been proved to be accurate and efficient [5].

*This research is supported by the European IST AMIGO (investigating AmI systems, to realize the full potential of home networking to improve people's lives) project (IST-004182).

ASE'05, November 7–11, 2005, Long Beach, California, USA.

[1]http://www.caida.org/tools/utilities/others/pathchar

Category	Dimension	Definition
Reliability	Availability	Probability=[0..1]
Performance	Latency	Service Time (in ms)
Cost	CPU Load	Percentage = [0..1]
	Memory	Percentage = [0..1]
	Bandwidth	Percentage = [0..1]
	Battery	Percentage = [0..1]
Security	Confidentiality	Boolean
	Integrity	Boolean
	Non-repudiation	Boolean
Transaction	Atomicity	Boolean
	Consistency	Boolean
	Isolation	Boolean
	Durability	boolean

Table 1: QoS dimensions in our base specification

3. SEMANTIC WEB SERVICE AND TASK DESCRIPTIONS

3.1 Service and Tasks Descriptions

OWL-S is a Web service ontology based on the Ontology Web Language (OWL), used to describe Web services properties and capabilities. This language describes Web services capabilities using three parts : the *service profile*, the *process model* and the *service grounding*. The service profile gives a high level description of a service and its provider, the process model describes the service's behavior as a process and the service grounding specifies the information necessary for service invocation. In our approach, each networked service is described as an OWL-S process model with QoS attributes on each involved atomic process (operation). These attributes are obtained from dynamic QoS measurement as described in section 2. On the other hand, the user task description is given in the form of an abstract OWL-S process with QoS requirements. The main difference between the user task's description and the services' descriptions is that contrary to the atomic processes involved in the services' processes, those involved in the user task process are not bound to any service, since services to be invoked are dynamically discovered. Finally, the non-functional requirements of the user task are expressed in the form of arithmetical constraints (e.g., $Availability > 50\%$) and have to be taken into account during the service composition. Our objective is to allow a user to perform a task on the fly, without any previous knowledge about the networking environment and to guarantee that the task will meet the requested QoS. Existing composition approaches based on OWL-S, use service profiles for matching services' advertised capabilities against users required capabilities, while the process model is used only for dynamic service invocation. However, we argue that the process model contains more information about the service capabilities than the service profile and may lead to a more precise matching. Therefore, in contrast to the approach implied by OWL-S, our matching between the task and the integrated services is based on the process model. Our composition approach is divided in two steps. First, semantic discovery of atomic processes delivers a set of services that provide atomic processes which are semantically equivalent with those of the user task. Second, QoS-aware process integration composes the selected services processes to produce the user task's process.

3.2 Modeling OWL-S Processes as finite state automata

Towards dynamic composition of services, we use formal modeling of OWL-S processes as finite state automata as described in [1]. This model defines mapping rules for translating an OWL-S process model to a finite state automaton. In this model, automata symbols correspond to the OWL-S atomic processes (the services operations) involved in the OWL-S process and a transition between two states is performed when an atomic process is executed. Each process involved in the OWL-S process model, is mapped to an automaton and linked together with the other automata in order to build the OWL-S process automaton. Both user tasks and networked services are modeled as finite state automata. However, the user task's automaton is complemented with additional information in some of its transitions, i.e., the probability for one transition to be selected. More precisely, a probability value is introduced in the case of a Repeat-While, Repeat-Until and Choice constructs. For the first two constructs (loops), the information added is the probability for the corresponding process to be executed once again. In the case of the Choice control construct, a probability is attached to each possible choice. This information is necessary to estimate the QoS of a composition on the basis of the user's histories. For example, if a composition involves a loop, the QoS of this composition depends on how many times the user will execute this loop. All these probabilities are evaluated based on histories and are updated each time the user task is executed.

In addition to the above probabilities that will be used to estimate the average QoS value of a service composition, some other information is needed to estimate the worst case value of QoS parameters. This information is attached to each loop construct in the task's process and gives the maximum number of times the loop can be carried out during the execution of a user task.

3.3 Evaluating the QoS of User Tasks

In our approach, as the task is abstract and does not refer to specific services, we need to extract the QoS formulae corresponding to each QoS metric. These formulae are extracted in advance and stored with the task's description. Then, during the composition, each time an abstract atomic process is replaced with a concrete one, these formulae are used to check the fulfillment of the task's QoS requirements. A number of research efforts propose reduction rules to compute the QoS of a workflow. We use the model in [2] to extract the formula of each QoS dimension, corresponding to the task's automaton structure. In this approach, a mathematical model is used to compute QoS for a given workflow process. More precisely, an algorithm repeatedly applies a set of reduction rules to a workflow until only one atomic node remains. This remaining node contains the QoS formula for each considered metric, corresponding to the workflow under analysis. We apply the reduction rules to OWL-S workflow constructs.

In our case, while evaluating the QoS of a service composition, we provide two estimations for each QoS dimension: (1) a history-based, probabilistic estimation; and (2) a pessimistic estimation. The former corresponds to an average estimation, while the latter corresponds to a worst case estimation. They are used to satisfy different user requirements, i.e., deterministic or probabilistic. For example, if the user

demands a deterministic QoS, our approach compares the requested QoS with the pessimistic estimation of the composite service. If the user requires an average QoS, the latter is compared against the probabilistic estimation.

4. QOS-AWARE, DYNAMIC TASKS COMPOSITION

The first step towards dynamic composition of user tasks is the semantic service discovery, which aims at selecting a set of services providing atomic processes that are semantically equivalent to the processes requested by the user task. This algorithm compares semantically the atomic processes of the user task with those of the networked services. Our semantic service discovery is based on the matching algorithm proposed by Paolucci et al. in [7]. This algorithm is used to match a requested service with a set of advertised ones. In our case we use this algorithm to match services' operations rather than services' capabilities.

4.1 QoS-Aware Process Integration

In this section, we present our QoS-aware process integration algorithm, which integrates the processes of the selected services to reconstruct the process of the target user task. To perform such an integration, we employ the finite state automata model that we have defined earlier. Thus, we consider the automaton representing the user task's process and the automata representing the selected services' processes. In a first stage, we connect the services' automata together to form a global automaton. The next stage is to parse each state of the task's automaton starting with the start state and following the automaton transitions. Simultaneously, a parsing of the global automaton is carried out in order to find for each state of the task's automaton an equivalent state of the global automaton. An equivalence is detected between a task's automaton state and a global automaton state when for each incoming operation of the former there is at least one semantically equivalent incoming operation of the latter. In addition to checking for each state the equivalence between incoming operations, a verification of the conformance to the QoS constraints of the user task is performed. This is done by using the QoS formulae that have been extracted from the task's automaton structure as described in Section 3.3. Thus, we start with the QoS formula for each QoS dimension, in which the QoS values of all operations are initialized with the best value (e.g., latency = 0, availability = 1). Then, each time we examine a service operation, we replace the corresponding best value in the formula of the considered dimension, with the real QoS value of the operation. This allows evaluating the values of all QoS dimensions if the current operation is selected. These values are then compared to the corresponding values required by the user task, and if the constraints are not met, the path in the global automaton that includes this operation is rejected.

Figure 1 gives an example of how we evaluate the aggregate QoS during the integration process. In this example, the abstract task automaton (left higher corner), which involves the operations op_1, op_2, op_3, op_4 and op_5, is going to be matched against the global automaton (right higher corner), which connects together the automata of services S_1, S_2, S_3 and S_4. These services have been selected after the semantic discovery stage, as their respective processes contain operations that are semantically equivalent with those of the abstract user task. Note that in this figure, for readability purposes, semantically equivalent operations are represented with the same names. The abstract user task has a single QoS requirement, which is $Latency < 6$. The QoS formula for average latency corresponding to the task's structure is given by: $Latency = \sigma_1(L_1+(\sigma_2*L_2+\sigma_3*L_3))+\sigma_4(L_4+L_5)$, where each L_i is the latency corresponding to the abstract operations op_i, and σ_i is the probability for op_i to be executed. Initially, the value of each L_i is assumed to be equal to 0. While browsing simultaneously the two automata, we can notice that when matching the operation op_1 of the user task against the equivalent operations $S_1.op_1$ and $S_2.op_1$ provided by the services S_1 and S_2, the latency in these two cases is respectively equal to: $0.8*(2+(0.3*0+0.7*0))+0.2*(0+0) = 1.6$ if $S_1.op_1$ is selected, and $0.8*(3+(0.3*0+0.7*0))+0.2*(0+0) = 2.4$ if $S_2.op_1$ is selected. In both cases the constraint $Latency < 6$ is met. Later, when matching the operations op_2 and op_3 of the user task, two paths are possible using the service S_1 alone, or using a composition of $S_2.op_1$ and S_4. However, the first choice gives a latency equal to: $0.8*(0.3*3+0.7*7)+0.2*(0+0) = 6.64 > 6$. Thus, this path is rejected, as it does not meet the QoS required by the user task. On the contrary, the second solution is kept. The complete resulting composition, which is a concrete realization of the abstract user task with available services, is shown in the left lower corner of the figure.

The proposed process integration algorithm gives a set of sub-automata from the global automaton that behave like the task's automaton and meet the QoS requirements of the user task. Once the set of possible compositions is given, the last stage is to choose the best among resulting compositions, on the basis of provided QoS. We evaluate every resulting service composition sc with the benefit function as proposed in [6]: Overall Benefit $= \sum_{i=1}^{n}(d_i * w_i)$, where d_i is the value of dimension d for the service composition sc, and w_i is the relative importance of the considered dimension. Note that different dimensions are in different units and are therefore normalized as in [6] to be comparable.

After selecting a composition scheme, an executable description of the user task that includes references to existing networked services is generated, and entered into an execution engine that executes this description by invoking the appropriate service operations.

5. PROTOTYPE IMPLEMENTATION AND EVALUATION

We have developed a prototype for evaluating the performance of our process integration algorithm. Whereas, our semantic service discovery is based on a semantic matching algorithm that have already been implemented and evaluated [7]. We have implemented our prototype in Java, on a Linux platform running on an Intel Pentium 4, 2.80 GHz CPU with 512 MB of memory. The performance of our process integration algorithm depends largely on the complexity of the task and the services candidate to the integration in terms of the number of involved operations. More specifically, the run time overhead of the algorithm is proportional to the number of possible (intermediate) composition paths assessed during the whole execution of the algorithm. We have carried out an experiment for evaluating the impact of the number of operations involved in the task's and

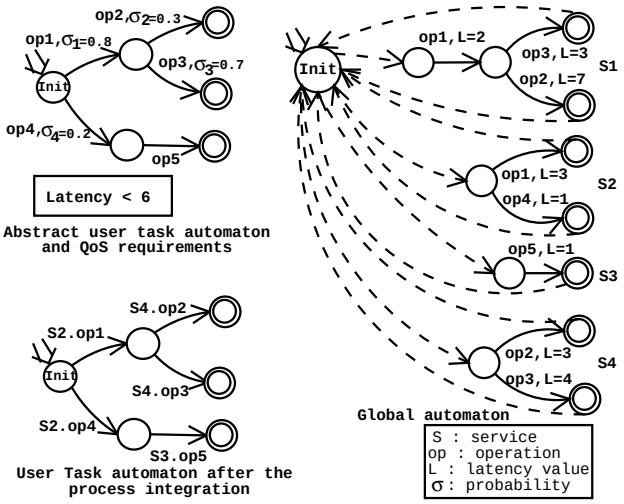

Figure 1: Example of the QoS-aware process integration

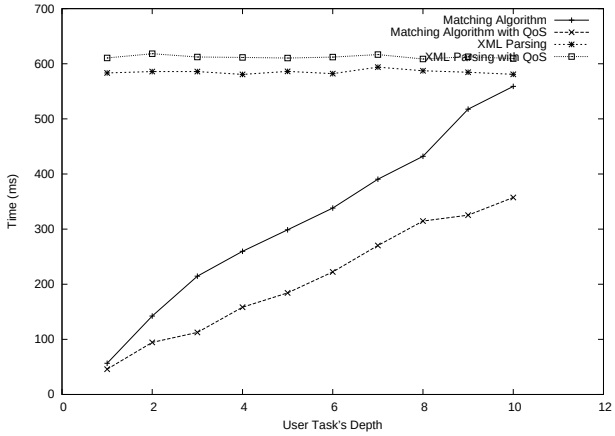

Figure 2: Performance of the Process Integration Algorithm with and without QoS

the services' processes on the performance of our algorithm. Furthermore, we have evaluated the impact of introducing QoS-awareness. Figure 2 shows the preliminary results of these experiments. In this figure the number of operations of the networked services is fixed to the extreme case equal to 100 semantically equivalent operations, while the task's number of operations is increasing from 1 to 10. Two results are reported in this figure: the performance of our algorithm with and without considering QoS. This figure shows an extreme scenario for our algorithm, as each operation of the user task is matched against 100 operations, and the resulting number of possible compositions is equal to: 100^{nb} in each case, where nb is the task's number of operations. We can see that even in the extreme case of 100^{10} possible compositions our algorithm takes less time than the XML parsing time of services' and task's descriptions. This figure shows also another important result, which is the impact of introducing QoS in our integration algorithm. This impact amounts to a small increase in the XML parsing time, which is due to the addition of XML tags for describing QoS, while at the same time to a considerable decrease of the execution time of our algorithm. This is attributed to the rejection of a number of paths that do not fulfill the QoS requirements of the user task during the integration.

6. CONCLUSION

The AmI vision foresees that the environment around us is populated with networked, both computing and input/output devices, that provide services. Our objective is to allow a user entering into an AmI environment to perform tasks, abstractly described on his/her device, by integrating on the fly available networked services. Our solution building on semantic Web services offers much more flexibility by enabling semantic matching of interfaces and dynamic integration of services' processes to perform the target user task. Furthermore, the computed service composition provides an estimated QoS that fulfills the initial QoS requirements of a user task. The distinctive feature of our solution is the ability to compose Web services that expose complex behaviors to realize a user task that itself has a complex behavior. Existing approaches in process-level matching generally consider that either the services or the task have a simple behavior, thus leading to simple integration solutions. In our case, we assume complex behaviors for both services and task described as OWL-S processes, and we propose a matching algorithm that attempts to integrate the services' processes to realize the user task. In order to deal with such level of complexity, our solution introduces an abstraction of OWL-S processes as finite state automata. A preliminary version of our matching algorithm was presented in [1]. The current version of the algorithm, which we have presented in this paper, introduces QoS-awareness, and a prototype implementation and evaluation. We have evaluated the performance of our matching algorithm with and without QoS-awareness. The results show that the proposed algorithm has a reasonable runtime overhead, and that the introduction of QoS constraints improves its performance. Our ongoing research effort includes the integration of the semantic discovery of operations into a scalable service discovery protocol, which is appropriate to AmI environments, optimizing at the same time the semantic matching performed during the discovery.

7. REFERENCES

[1] S. Ben Mokhtar, N. Georgantas, and V. Issarny. Ad hoc composition of user tasks in pervasive computing environments. In *Proceedings of SC'05. LNCS 3628.*

[2] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and Web service processes. *Journal of Web Semantic*, 2004.

[3] H. V. Dijk, K. Langendoen, and H. Sips. ARC: a bottom-up approach to negotiated QoS. In *Proceedings of WMCSA'00.*

[4] J. Flinn, S. Y. Park, and M. Satyanarayanan. Balancing performance, energy, and quality in pervasive computing. In *Proceedings of ICDCS'02.*

[5] S. Gurun, C. Krintz, and R. Wolski. NWSLite: a light-weight prediction utility for mobile devices. In *Proceedings of ACM MobiSys'04.*

[6] J. Liu and V. Issarry. Qos-aware service location in mobile ad-hoc networks. In *Proceedings of MDM'04.*

[7] K. S. N. Srinivasan, M. Paolucci. An efficient algorithm for owl-s based semantic search in UDDI. In *Proceedings of SWSWPC'04.*