Flexible Provisioning of Semantic Web Service Workflows using a QoS Ontology

Sebastian Stein and **Terry R. Payne** and **Nicholas R. Jennings**
University of Southampton
Southampton, SO17 1BJ, UK
{ss04r,trp,nrj}@ecs.soton.ac.uk

Abstract

Semantic Web services allow applications to automatically discover and provision distributed services at runtime. However, when such services are offered by autonomous providers, as is common in large distributed systems, their behaviour is inherently non-deterministic and unreliable. To address this problem, we describe an OWL-S extension that allows services to be annotated with quantitative performance measures, and outline a flexible, decision-theoretic algorithm that uses this knowledge to provision services as part of complex workflows.

1 Introduction

Due to the proliferation and ubiquity of the Internet, modern computer systems are becoming increasingly distributed. This growing connectivity allows organisations to share expensive computing resources, to automate and outsource business processes, and to offer their services to a worldwide audience. In this context, Semantic Web services are emerging as a key technology to describe and discover distributed software components at run-time [McIlraith *et al.*, 2001].

Current work in this area has so far concentrated largely on logical formalisms that assume truthful service descriptions and deterministic behaviour, and has therefore neglected the fact that services in distributed systems are inherently unreliable and non-deterministic. However, such uncertainty should be considered, due to the open and dynamic nature of the Internet, where network failures, remote software bugs, transmission delays and competition over limited resources are an unavoidable feature of the environment, and where services are offered by autonomous and self-interested agents.

To address this problem of service uncertainty, we focus on the *provisioning* of Semantic Web services, which has so far been largely overlooked by the current literature. In short, this is the process of assigning particular service instances to the constituent tasks of abstract workflows after candidate services have been identified by a semantic matchmaker. Whilst the matchmaking stage concentrates on matching functional service adverts to abstract task templates, provisioning uses advertised or observed quality-of-service measures to allocate instances in an appropriate manner. Thus, it is possible during provisioning to make predictions about the overall performance of the workflow, to identify particularly failure-prone tasks and to mitigate the effects of such tasks by provisioning services appropriately. In particular, by explicitly provisioning services, we are able to design a decision-theoretic algorithm that aims to maximise the expected utility of a service-consuming agent.

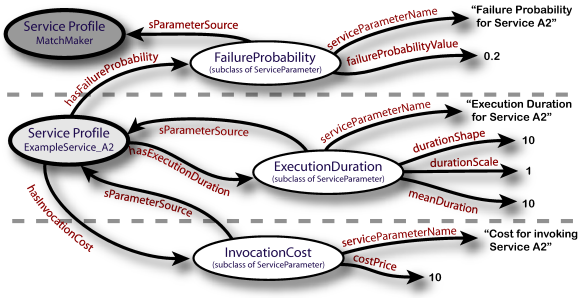

Figure 1: Quality-of-Service Ontology

2 Quality-of-Service Ontology

Current Semantic Web service ontologies such as OWL-S[1] provide rich models for representing the functional descriptions for services, but provide simple, high-level classes for representing the non-functional information that is vital for making decisions when services behave non-deterministically. To address this, we present an extension to OWL-S for expressing a subset of non-fuctional parameters that support effective service provisioning, and illustrate how these can affect the performance of instansiated workflows within uncertain environments.

Figure 1 illustrates our ontology extension along with example instances[2]. This extension consists of several new classes, each of which subclass *ServiceParameter*, and consequently define subproperties of *serviceParameter* and *sParameter*. Three new classes have been defined to date, each providing quantative (i.e. *Datatype*) properties:

FailureProbability: Defines the failure probability of a service in the range [0..1], i.e., what is the likelihood that the service might fail to deliver the desired service.

ExecutionDuration: Defines the execution duration of a service. This is modelled using a probability density func-

[1] See `http://www.daml.org/services/owl-s/`

[2] The full ontology and further examples are available at `http://www.ecs.soton.ac.uk/~ss04r/provisioning/`

tion, as service durations can vary randomly (e.g., due to network delays or high workloads). In this ontology, a gamma distribution is modelled through specifying three *Datatype* properties:

```
<owl:DatatypeProperty rdf:ID="durationShape">
  <owl:subPropertyOf rdf:resource=
                     "../Profile.owl#sParameter"/>
  <rdfs:domain rdf:resource="#ExecutionDuration"/>
  <rdfs:range rdf:resource=
       "http://www.w3.org/2001/XMLSchema#double"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="durationScale">
  ...
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="meanDuration">
  ...
</owl:DatatypeProperty>
```

InvocationCost: Defines the cost of invoking a service. This usually represents a financial remuneration for the service, but could also quantify the effort and required bandwidth of invoking the service.

Although some parameters (such as *InvocationCost*) can be specified by the service provider itself, others (e.g. *FailureProbability*) may lack credibility unless provided by a third party. A third property, *sParameterSource*, determines the source (or provenance) of the service parameter values, and thus may be used by a requester to determine whether the source (and thus the parameters) can be trusted.

3 Flexible Provisioning

Given this ontology and associated performance information, it is now possible to anticipate and deal with uncertain service providers. In particular, we suggest two key techniques:

Parallel Provisioning: To increase the probability of success, n services that match the same abstract description can be provisioned in parallel for a single task. The task will then succeed as long as at least one service completes successfully.

Serial Provisioning: Services can also be re-provisioned when it becomes obvious that they have failed. As failure may not be communicated explicitly by the providers, this process involves provisioning an initial set of services, and then waiting for some time w, after which a new set of services is provisioned for the same task (and so on).

Now, the appropriate choice of n and w for each task depends on the performance characteristics of the available services, as given by our QoS ontology. For example, if services are cheap (low *InvocationCost*) and unreliable (high *FailureProbability*), a high value for n might be chosen. If services generally take a long time to complete (as given by the *ExecutionDuration*), then a high w would reflect this.

Because finding the right level of parallelism and waiting times is non-trivial to perform manually, and because it depends on the consumer's valuation of the workflow and associated time-constraints, we automate this choice by employing decision theory. To this end, we attach a utility function to each workflow, in order to represent the value of completing the workflow after some time t. As provisioning services optimally (i.e., maximising the expected utility) is intractable, we then use a hill-climbing algorithm that uses a heuristic estimation of the true expected utility and explores a small subset of the decision space by iteratively modifying a candidate solution [Stein *et al.*, 2006].

4 Empirical Evaluation

We implemented a simulation of a service-oriented system and used this to empirically evaluate the benefit of employing our QoS ontology when provisioning services. In our experiments, we measured the average profit of a consumer agent as the difference between the profit from completing a workflow (given by the utility function) and the total cost (incurred when invoking services). To evaluate the utility of considering QoS parameters with our provisioning approach, we compared it to a simple strategy which does not use the QoS parameters defined within the ontology (Figure 1), and thus provisions only single, arbitrary services for each constituent task of its workflows.

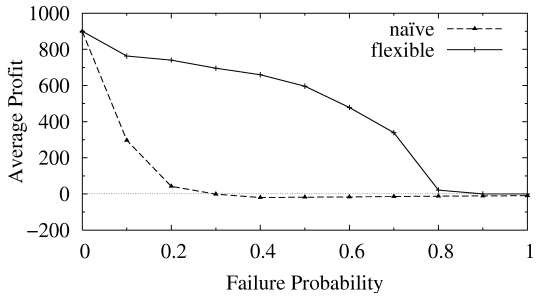

Figure 2: Average Profit of Flexible Provisioning

Figure 2 shows the profit of our decision-theoretic approach (*"flexible"*) over the benchmark (*"naïve"*) as the failure probability of services in the system increases[3]. In this particular scenario, a workflow consisted of 10 sequential tasks, each of which could be satisfied by 1,000 homogeneous services with a cost of 10 units and a random duration distributed according to a gamma distribution with shape $k = 2$ and scale $\theta = 10$. The consumer was awarded a utility payoff of 1,000 if the workflow was completed within 400 time steps, after which a cumulative penalty of 10 was applied per time step until a zero-payoff was reached.

The results show clearly that our approach of using a QoS ontology to provision services outperforms the naïve strategy (when averaged over all failure probabilities, it obtains a 350% improvement in average profit). We confirmed this trend in a variety of other experimental settings, including highly parallel workflows and heterogeneous service types.

References

[McIlraith *et al.*, 2001] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[Stein *et al.*, 2006] Sebastian Stein, Nicholas R. Jennings, and Terry R. Payne. Flexible provisioning of service workflows. In *Proceedings of 17th European Conference on Artificial Intelligence (ECAI-06)*, pages 295–299. IOS Press, 2006.

[3] All results are averaged over 1,000 experimental runs.