Flexible Provisioning of Service Workflows

Sebastian Stein and **Nicholas R. Jennings** and **Terry R. Payne**[1]

Abstract. Service-oriented computing is a promising paradigm for highly distributed and complex computer systems. In such systems, services are offered by provider agents over a computer network and automatically discovered and provisioned by consumer agents that need particular resources or behaviours for their workflows. However, in open systems where there are significant degrees of uncertainty and dynamism, and where the agents are self-interested, the provisioning of these services needs to be performed in a more flexible way than has hitherto been considered. To this end, we devise a number of heuristics that vary provisioning according to the predicted performance of provider agents. We then empirically benchmark our algorithms and show that they lead to a 350% improvement in average utility, while successfully completing 5–6 times as many workflows as current approaches.

1 INTRODUCTION

Computer systems are becoming increasingly complex, distributed and open in nature. In so doing, they pose new challenges to contemporary software engineering techniques, as applications need to deal with many widely distributed and dynamic resources. In this context, service-oriented computing has gained popularity as an appropriate paradigm to build such systems [3]. In this approach, resources are offered as services, which are advertised to consumer applications using computer-readable descriptions and then provisioned dynamically when needed, often as part of complex workflows [2].

A defining feature of these distributed systems is that participants are usually heterogeneous, self-interested agents that act autonomously in accordance with their own private goals [4]. As such, agents choose actions that maximise their own welfare, and so they cannot be assumed to honour every request, to be available whenever they are needed or even to report their capabilities and status truthfully. Additionally, network problems, competition for resources or even software bugs exacerbate the potential for service failures. Thus, when interacting with service providing agents and making real investments that depend on their performance (whether in terms of money, time or other resources), dealing with service failures becomes a vital issue to consider.

Against this background, we believe that flexible service provisioning (i.e., selecting and allocating service providers to specific tasks) is the key to handling and controlling service failures. It allows the consumer to dynamically select service providers based on their performance characteristics and provision replacement providers when services fail. Specifically, provisioning services in the context of a workflow enables a consumer agent to identify particularly failure-prone tasks and invest additional resources in them. To date, however, this provisioning process has received comparatively little attention. In particular, the fact that a type of service might be offered by multiple providers is seldom explored, and most research is concerned with finding and invoking the first matching service that helps fulfil a given goal [6]. Such a naïve approach means that a single service failure will result in the failure of the whole workflow, which is highly undesirable, especially when success is critical to the interests of the consumer agent.

When it has been considered, this problem is usually addressed reactively by dynamic replanning in case of failure [5], but this method is computationally expensive and can lead to long delays when there are particularly unreliable services. In [1], the authors take a more proactive approach by provisioning services that maximise an expected utility function. However, they again provision only single providers, which makes workflows vulnerable to service failures.

To address these shortcomings, we investigate the process of provisioning service workflows in environments where service success is not generally guaranteed. We show that the current approach of provisioning single providers is insufficient in such environments, and then advance the state of the art by developing several strategies that deal proactively with unreliable providers. This is achieved by provisioning several redundant service providers for a single task in order to reduce the associated failure probability, and by provisioning new providers to those tasks that appear to have failed. We also develop a novel heuristic that provisions workflows in a flexible manner depending on the characteristics of the service providers. Specifically, our heuristic provisions more providers to tasks that are particularly likely to fail, while relying on fewer providers when success is more certain. In order to verify our approach, we report our empirical results that show the value of flexible service provisioning over the naïve approach. In particular, we show that our heuristic successfully completes over 90% of its workflows in most environments where current approaches complete none. Furthermore, the results indicate that our heuristic achieves a 350% improvement in average utility over the naïve strategy.

The remainder of this paper is organised as follows. In the next section we present two simple provisioning strategies, followed by an extended strategy that provisions services in a flexible manner. In Section 3 we describe our experimental testbed and present empirical results for our strategies. Section 4 concludes.

2 PROVISIONING STRATEGIES

In this section, we outline the type of dynamism and uncertainty that service providers display in the complex systems we consider. This is followed by an overview of the kinds of workflows and associated rewards that service consumers typically face. We then present two strategies (*parallel* and *serial*) that provision providers redundantly, but in an inflexible manner, in order to increase their chance of success. Finally, we outline a flexible strategy that provisions providers depending on the agent's assessment of its situation.

[1] School of Electronics and Computer Science, University of Southampton, United Kingdom, {ss04r,nrj,trp}@ecs.soton.ac.uk

2.1 Modelling Service-Oriented Systems

As discussed in Section 1, the inherently uncertain behaviour of autonomous service providers can pose serious threats to the goals of a service consumer. Providers are generally not guaranteed to execute services successfully, and even when they do, the time of completion may be influenced by network traffic, other commitments the provider has made and the hardware a service is executed on. Hence, we model services probabilistically and we assume that some information has already been learnt about the performance characteristics of the service providers for a given task, including their average failure probability and a distribution function for the duration of a successful service execution.

In service-oriented systems, service consumers often face large workflows of inter-dependent tasks. For the purpose of this paper, we represent such workflows as directed, acyclic graphs with the nodes N being tasks and the edges E defining the dependencies between tasks (i.e., an edge $(t_1 \mapsto t_2)$ means that task t_1 has to complete successfully before t_2 can be started). A workflow is only considered complete when all its constituent tasks have been completed.

Furthermore, to evaluate our strategies, we define a utility function $u(t)$ to capture the value of finishing a workflow at time t. We chose to represent this by a maximum utility $u_{\max}$, awarded for completion, a deadline d, and a penalty charge p, to be deducted from the final utility for every time step a workflow is late. Formally, we write:

$$u(t) = \begin{cases} u_{\max} & \text{if } t \leq d \\ u_{\max} - p(t-d) & \text{if } t > d \text{ and } t < d + u_{\max}/p \\ 0 & \text{if } t \geq d + u_{\max}/p \end{cases} \tag{1}$$

As is common in contemporary frameworks [3], services in our model are invoked *on demand*. Specifically, a consumer requests the execution of a provisioned service when the associated task becomes available. At that time, the consumer incurs a cost (e.g., financial remuneration or communication cost), which is assumed uniform among the providers of a certain task. Thus, the overall profit of a workflow execution is the difference of the utility of finishing the workflow (or 0 if unsuccessful) and its total cost.

Having outlined our environment and basic assumptions, we now continue to develop several provisioning strategies. As discussed, the aim of these is to improve upon the currently predominant strategy of provisioning a single, randomly chosen service provider for each task in a workflow — a strategy that we refer to as the ***naïve*** strategy.

2.2 Parallel Service Provisioning

Recognising the ease of discovering substitute services in service-oriented systems, our first strategy uses redundant service provisioning to control the effect of unreliable providers. In general, if the success probability of a randomly chosen provider is S_1, then provisioning n providers for a task results in a success probability S_n:

$$S_n = 1 - (1 - S_1)^n \tag{2}$$

When $S_1 > 0$, then $\lim_{n\to\infty} S_n = 1$, which implies that it is possible to increase the probability of success to an arbitrarily high amount as long as there are sufficient providers in the system. However, as more providers are provisioned, the total cost incurred rises linearly with n.

This result leads us to our first strategy, ***parallel(n)***, which always provisions exactly n randomly chosen members of the set of available providers for a given task. The strategy *parallel(1)*, here, is a special case that is equivalent to the naïve strategy.

2.3 Serial Service Provisioning

An alternative approach to relying on parallel provisioning of providers to increase the probability of success, is to re-provision services when it becomes apparent that a particular provider has failed. In this case, the consumer first provisions a single provider and, after invocation, waits for some time. If the provider has not been successful, the consumer tries a different provider and so on. However, as providers cannot generally be assumed to notify the consumer of failure and because they have non-deterministic duration times, the consumer has to choose an appropriate waiting period. This period should give the provider a reasonable time to finish, but should not waste unnecessary time when the service has already failed[2].

With this in mind, let $T(w)$ be the probability that a randomly chosen service provider successfully completes within the waiting time w (note that $T(w) \leq S_1$). Then the success probability of serial provisioning with n available providers is:

$$T_n(w) = 1 - (1 - T(w))^n \tag{3}$$

This is usually less than the success probability of provisioning the same number of providers in parallel and the average time taken will also be higher for serial provisioning because of the additional waiting time that is introduced. On the other hand, the average cost drops, because costs are only incurred at the time of invocation.

This leads us to our second strategy, ***serial(w)***, which always provisions exactly one randomly chosen member of the set of available providers. After a waiting period of w time units, if no success has been registered yet and if there are still more providers, the agent re-provisions a new, randomly chosen provider. A special case of this strategy, *serial(∞)*, is equivalent to the naïve strategy.

2.4 Flexible Service Provisioning

The strategies discussed so far provision services in an inflexible manner, as they always select the same number of providers for each task. This approach is insufficient in most scenarios, however, because some services may benefit from being provisioned redundantly, while others have a high degree of certainty and so need not be provisioned in this manner. Furthermore, it is not clear how to choose n and w in the above strategies so as to balance the associated cost and maximise the expected utility of a workflow, and it is desirable to automate this decision.

To address these shortcomings, in this section we develop a ***flexible*** strategy that combines the above two strategies, *parallel(n)* and *serial(w)*, by automatically finding values for the number of parallel invocations (n_i) and waiting time (w_i) for every task t_i in a given workflow. To focus on the basic problem, we assume that this allocation is made once and then used non-adaptively for the entire execution of the workflow[3]. We also assume that the consumer is risk-neutral — that is, we want our algorithm to choose values n_i and w_i in order to maximise the consumer's long-term expected profit.

Now, many optimisation approaches exist in the literature, but due to the complex nature of this particular problem, which includes the

[2] When re-provisioning services, we assume that all previously provisioned services are subsequently ignored (even if they had succeeded at a later time). This assumption draws a clear distinction between serial and parallel provisioning techniques, but does not fundamentally alter our results.

[3] This simplifies the problem and helps us verify that our algorithm makes good initial predictions even without receiving ongoing feedback during a workflow. However, in future work we will use such information to update the agent's service provisions dynamically.

summation of random variables (the service durations), integer variables and a non-linear objective function, we have decided to use a local search technique that approximates the expected profit using a heuristic function, and searches for a good allocation rather than an optimal one.

Specifically, our local search algorithm begins with a random allocation and then applies steepest ascent hill climbing [7] to gradually improve the allocation until a maximum is found. The heuristic function we use is based on the utility function u and we use it to calculate the estimated profit $\tilde{u}$:

$$\tilde{u} = p \cdot u(\tilde{t}) - \bar{c} \tag{4}$$

where p is the probability of finishing all tasks at some point in time, $\tilde{t}$ is the estimated completion time for the workflow, and $\bar{c}$ is the expected cost.

In the following, we explain how p, $\tilde{t}$ and $\bar{c}$ are calculated, starting with appropriate calculations for each individual task in Section 2.4.1 and then extending this to the whole workflow in Section 2.4.2.

2.4.1 Local Task Calculations

We are interested in three performance measures for each task t_i in the workflow — the overall success probability of the task (s), its expected cost ($\bar{c}$) and its expected duration ($\bar{d}$). These are calculated using the information that the consumer has about each task and the relevant providers:

- p is the number of providers provisioned in parallel for the task.
- w is the waiting time before a new set of providers is provisioned.
- a is the number of available providers.
- f is the average failure probability of a provider.
- c is the cost of a provider.
- D is the cumulative density function for a single service duration.

First, we calculate the probability of success s as in equation 3, but noting that $T = (1 - f) \cdot D(w)$ and using the total number of available providers:

$$s = 1 - (1 - (1 - f) \cdot D(w))^a \tag{5}$$

In showing how to calculate the total cost, we assume that $a \bmod p = 0$ (i.e., that we can invoke up to $m = a/p$ sets of p providers with no remaining providers at the end[4]). Each invoked set of providers then has a probability of success $\hat{s} = 1 - (1 - (1 - f) \cdot D(w))^p$ and an associated cost cp. The consumer is guaranteed to pay cp at least once, and may pay again if the previous invocation was unsuccessful with probability $\hat{f} = (1 - \hat{s})$. Using this, and assuming that $\hat{f} < 1$, we can write the expected cost $\bar{c}$ as:

$$\bar{c} = cp \sum_{k=0}^{m-1} \hat{f}^k = cp \frac{1 - \hat{f}^m}{1 - \hat{f}} \tag{6}$$

Using the same assumptions and treating the simpler case with $a \bmod p = 0$, we now investigate how to calculate the expected duration of a task, $\bar{d}$. We define this to be the mean time until the first provider carries out the task successfully, conditional on an overall success (i.e., at least one provider is successful).

First, we define μ to be the mean duration of a single successful invocation of p providers (also conditional on overall success). Then we follow a similar technique for calculating $\bar{d}$ as we did for $\bar{c}$. We consider all possible outcomes for the invocation by calculating the expected duration when the task is completed after exactly k failed invocations as well as the associated probability. Then we multiply these durations with their probabilities and sum them to get the overall expected duration.

[4] Due to space restrictions in this paper, we omit the general case.

More formally, if a task succeeds after exactly k unsuccessful invocations, the expected duration, $\bar{d}_k$, is the expected duration for a single invocation added to k waiting time durations w for the previously failed invocations:

$$\bar{d}_k = \mu + kw \tag{7}$$

The probability of the task completing after the kth attempt is:

$$s_k = \hat{f}^k (1 - \hat{f}) \tag{8}$$

With this, we calculate the overall expected duration, conditional on at least one provider being successful (assuming $\hat{f} < 1$):

$$\begin{aligned} \bar{d} &= \frac{1}{s} \cdot \sum_{k=0}^{m-1} \bar{d}_k s_k \\ &= \frac{1}{s} \cdot \sum_{k=0}^{m-1} \Big((\mu + kw) \cdot \hat{f}^k (1 - \hat{f}) \Big) \\ &= \frac{1}{s} \cdot \left(\mu (1 - \hat{f}^m) + w \frac{\hat{f} - m\hat{f}^m + (m-1)\hat{f}^{m+1}}{1 - \hat{f}} \right) \end{aligned} \tag{9}$$

We have now shown how to calculate various performance characteristics of a single task. In the following section, we explain how this is extended to calculate the overall heuristic function for an allocation over the whole workflow.

2.4.2 Global Workflow Calculations

Let s_i, $\bar{c}_i$ and $\bar{d}_i$ be the success probability, the expected cost and the expected duration of task t_i. With this information for each task, we are now interested in calculating the overall probability of success s, the estimated completion time of the workflow $\tilde{t}$ and the total expected cost $\bar{c}$.

The overall probability of success is simply the product of all s_i:

$$s = \prod_{\{i | t_i \in N\}} s_i \tag{10}$$

The expected total cost is the sum of all task costs, each multiplied by the respective success probabilities of their predecessors in the workflow (where r_i is the probability that task t_i is ever reached):

$$\bar{c} = \sum_{\{i | t_i \in N\}} r_i c_i \tag{11}$$

$$r_i = \begin{cases} 1 & \text{if } \forall t_j \cdot ((t_j \mapsto t_i) \notin E) \\ \prod_{\{j | (t_j \mapsto t_i) \in E\}} s_j & \text{otherwise} \end{cases} \tag{12}$$

Finally, we approximate the overall time $\tilde{t}$ using the length of the critical path in the workflow. This is the length of the longest path from any node with no predecessors to any node with no successors, using the expected durations $\bar{d}_i$ as weights attached to the nodes. Such an approach will normally underestimate the true expected duration, because it focusses only on one path, ignoring the possibility that other services outside this path may take longer. However, it provides a fast approximation and has been used widely in project management and operations research [8]. Formally, we let $P = \{t_i \mid t_i \text{ is on the critical path}\}$. This gives us:

$$\tilde{t} = \sum_{\{i|t_i \in P\}} \bar{d}_i \quad (13)$$

Using these values and the heuristic function given in equation 4, it is now possible to use steepest ascent hill-climbing to find a good allocation of providers. In practice, we found it useful to perform this in two iterations — once using a modified utility function $\tilde{u}$, which is a linear version of u, giving a higher reward than usual for finishing early and a larger loss for finishing late, and then again using the normal utility function u. This approach allows the algorithm to escape from a common local maximum where the agent decides to concede, allocate minimal resources to the tasks and hence incur a low net loss, but also a very low probability of success. Furthermore, we could generally increase performance by adding a constant amount of extra time to $\tilde{t}$ to account for the error in the prediction. In all our experiments we set this to 20% of the workflow deadline, as this produces good results in a variety of experimental settings.

3 EMPIRICAL EVALUATION

As stated in Section 1, the aim of our work is to deal effectively with unreliable service providers when provisioning workflows. To this end, in this section we empirically compare our proposed strategies to the currently predominant naïve approach. In particular, we investigate the average utility gained by all strategies, as well as the average proportion of successfully completed workflows. In the remainder of this section, we describe our experimental testbed and methodology, followed by the results.

3.1 Experimental Testbed

In order to analyse our strategies empirically, we developed a simulation of a simple service-oriented system. In this simulation, we generate large numbers of random workflows and measure the performance of our strategies by recording the percentage of workflows that failed (where t time steps had elapsed, so that $u(t) \leq 0$) and succeeded (where all tasks were completed within t time steps, so that $u(t) > 0$). We also measure the average profit of a single service consuming agent.

Our simulation is discrete, notifying the consumer of any successful service execution once every integer time step, at which point new services are also invoked according to the consumer's provisioning strategy. While the consumer is notified in case of service failure, no information is given when a provider fails.

To simplify the analysis and because our approach does not deal directly with differentiating between individual providers at this time, we examined environments with homogeneous service providers (i.e., all providers share the same success probability and duration distributions).

For the data presented in this section, we used workflows consisting of 10 tasks in a strict series (i.e., without parallel tasks, because this allowed us to verify some results analytically), we assumed that there were 1,000 providers for every task with each provider having a cost of 10 and a gamma distribution with shape $k = 2$ and scale $\theta = 10$ as the probability distribution of the service duration. We set a deadline of 400 time units for the workflow, an associated maximum utility of 1,000 and a penalty of 10 per time unit. We also performed similar experiments in a variety of environments, including heterogeneous and parallel tasks, and observed the same broad trends that are presented in the following section.

To prove the statistical significance of our results, we averaged data over 1,000 test runs and performed an analysis of variance (ANOVA) where appropriate to test whether the strategies we tested produced significantly different results. When this was the case, we carried out pairwise comparisons using the least significant difference (LSD) test. Thus all results reported in this paper are statistically significant ($p = 0.001$).

3.2 Experimental Results

In our first experiment, we compared the performance of strategy *parallel(n)*[5] with the naïve approach in environments where service providers have a varying probability of failure (see Figure 1). From this, it is clear that there is a considerable difference in performance between the different strategies — the average profit gained by the naïve strategy falls dramatically as soon as failures are introduced into the system. In this case, the average utility of provisioning single providers falls to below 0 when the failure probability of providers is only 0.3. A statistical analysis reveals that the naïve strategy dominates the other two when there is no uncertainty in the system. However, as soon as the failure probability is raised to 0.1, *parallel(2)* begins to dominate the other strategies. Between 0.3 and 0.6 *parallel(6)* then becomes the dominant strategy as increased service redundancy leads to a higher probability of success. Above this, the parallel strategies do not yield better results than the naïve strategy as they also begin to fail in most cases.

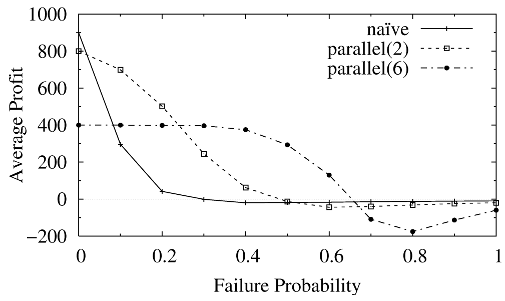

Figure 1. Effect of provisioning different numbers of providers in parallel

Summarising these trends, it is obvious that redundant provisioning yields a considerable improvement over the naïve approach in a range of environments. For example, when the failure probability is 0.2, provisioning two providers results in an almost 1,100% improvement in average profit over the naïve strategy. However, no redundant strategy dominates the other and both eventually make losses when the probability of failure increases to such an extent that the chosen redundancy levels do not suffice to ensure success.

We carried out a similar experiment to verify the advantage of serial provisioning over the naïve strategy (see Figure 2). Here, again, there is a marked improvement over the naïve strategy for failure probabilities up to and including 0.5. This improvement is due to the fact that serial provisioning responds to failures as they occur, while only paying for additional services when necessary. However, as the failure probability rises, this strategy begins to miss its deadlines and hence incurs increasingly large losses.

Finally, to show how the *flexible* strategy compares against the naïve provisioning approach and inflexible redundancy, Figure 3 shows our experimental data (again, using the same experimental variables). The top graph shows the percentage of workflows that

[5] Here, we arbitrarily chose $n = 2$ and $n = 6$ as representative of the general trends displayed by the strategy as more providers are provisioned.

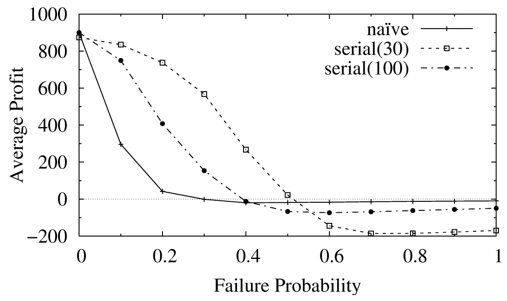

Figure 2. Effect of different amounts of waiting times for re-provisioning

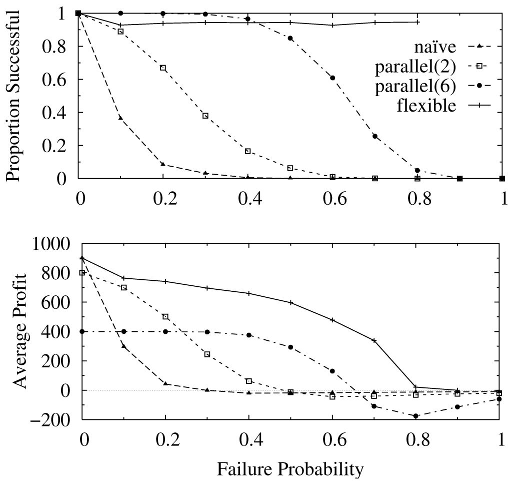

Figure 3. Performance of flexible strategy

succeeded out of all the ones generated. Here, we see that our heuristic approach initially performs slightly worse than *parallel(6)* (but always significantly better than the naïve approach). This is due to our technique of using the critical path of the workflow as an estimate for the total time taken. This technique is usually too optimistic and might result in under-provisioned tasks. However, the graph clearly shows that the flexible technique still achieves a success-rate of over 90% and, more importantly, maintains this up to a failure probability of 0.8, by which all other approaches have large failure rates. When 0.9 is reached, the strategy begins to ignore all workflows, because it cannot find a feasible allocation to offer a positive return. On the lower graph, we show the average utility that is gained by the same strategies. Here, it is clear that the flexible approach performs better than any of the other strategies. This is due to its utility prediction mechanism and the fact that it can make choices separately for the various tasks in the workflow. This flexibility allows the strategy to provision more providers for latter parts of the workflow, where success becomes more critical as a higher investment has already been made. The flexible approach also combines the benefits of the other strategies, allowing the agent to choose between parallel (e.g., when there is little time) and serial provisioning (e.g., when the agent can afford the extra waiting time) or a mixture of the two. Although performance degrades as providers become more failure-prone, flexible provisioning retains a relatively high average utility when all other strategies start to make a loss. Furthermore, the strategy avoids making a loss due to its prediction mechanism, which ignores a workflow when it seems infeasible.

To conclude, Table 1 summarises the performance of some representative strategies, averaged over all environments that we tested (using the same data as in Figures 1–3). These results highlight the benefits of our strategies, and show that our flexible strategy by far outperforms the naïve approach. In particular, we achieve an improvement of approximately 350% in mean profit and successfully complete 76-78% of all workflows.

Table 1. Summary of results with 95% confidence intervals

Strategy	Mean profit	Profit vs naïve	Success rate
naïve	103.09 ± 5.30	1	0.13 ± 0.01
parallel(6)	175.87 ± 5.79	1.71 ± 0.20	0.61 ± 0.01
serial(30)	221.91 ± 7.92	2.15 ± 0.26	0.44 ± 0.01
flexible	471.99 ± 8.71	4.58 ± 0.49	0.77 ± 0.01

4 CONCLUSIONS

In this paper, we addressed the problem of dealing with unreliable service providers in complex systems. To this end, we developed a novel algorithm for provisioning workflows in a flexible manner and showed that this algorithm achieves a high success probability in uncertain environments and that it outperforms the current, naïve strategy of provisioning single providers. This work is particularly relevant for distributed applications where several services are composed by a consumer agent, and where these services are offered by autonomous agents whose success cannot be guaranteed. Important application domains for our approach include scientific data processing workflows and distributed business applications, where services are sourced from across departments or organisations.

In future work, we plan to extend our approach to cover more heterogeneous environments, where service providers differ significantly in terms of cost and reliability within the same service type and where such qualities are affected dynamically by system-wide competition and resource availability. Furthermore, we plan to examine in more detail the computational cost of provisioning and how this can be balanced in a flexible manner with the need to make quick decisions in dynamic environments.

ACKNOWLEDGEMENTS

This work was funded by the Engineering and Physical Sciences Research Council (EPSRC) and a BAE Systems studentship.

REFERENCES

[1] J. Collins, C. Bilot, M. Gini, and B. Mobasher, ‘Decision Processes in Agent-Based Automated Contracting’, *IEEE Internet Computing*, **5**(2), 61–72, (2001).

[2] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda, ‘Mapping Abstract Complex Workflows onto Grid Environments’, *Journal of Grid Computing*, **1**(1), 25 – 39, (2003).

[3] M. N. Huhns and M. P. Singh, ‘Service-Oriented Computing: Key Concepts and Principles’, *IEEE Internet Computing*, **9**(1), 75–81, (2005).

[4] N. R. Jennings, ‘On Agent-Based Software Engineering’, *Artificial Intelligence*, **117**(2), 277–296, (2000).

[5] M. Klusch, A. Gerber, and M. Schmidt, ‘Semantic Web Service Composition Planning with OWLS-XPlan’, in *Proceedings of the 1st International AAAI Fall Symposium on Agents and the Semantic Web*, (2005).

[6] S. A. McIlraith and T. C. Son, ‘Adapting Golog for Composition of Semantic Web Services’, in *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002)*, pp. 482–493, Toulouse, France, (2002).

[7] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Englewood Cliffs, NJ, USA, 2nd edn., 2003.

[8] W. L. Winston, *Operations Research: Applications and Algorithms*, Wadsworth Publishing Company, 3rd edn., 1997.