Realising Personalised Web Service Composition through Adaptive Replanning

Steffen Higel, David Lewis, Vincent Wade

Knowledge and Data Engineering Group, Department of Computer Science, Trinity College Dublin, Ireland

{higels, delewis, vwade}@cs.tcd.ie

Abstract. The emergence of fully-automated Web service composition as a potential facilitator of both eBusiness and ambient or ubiquitous computing is to be welcomed. However this emergence has exposed the need for flexibility and adaptivity due to the fundamentally unreliable nature of the networks and infrastructure on which the component services rely. Furthermore, a key to driving forward acceptance and adoption of this growing set of technologies is the improvement of the user's overall experience therewith. Our experimentation has proven that it is quite possible to generate inflexible and only partially adaptive service compositions using out of the box A.I. planners. Because modifying the planner is beyond the scope of our research, we seek to use methods of pre-processing and post-analysis to enable AI planners to produce adaptive compositions. In this paper, the current state of our research is presented along with a proposed direction for improving the reconciliation of user needs with the available services.

1 Introduction

The composition of software components into larger pieces of useful software is a remarkably simple concept. We feed the outputs of one component into the inputs of one or more others in the anticipation that the net effect will be close to what we want to achieve. When coupled with semantics supported by ontologies, the mature manageability and audit ability found in work-flow and the power of A.I. planning, it should become possible to automatically generate robust and personalised compositions at near run-time in response to a user's needs.

Web services and their associated technologies have become an enabler of "loosely coupled, dynamically bound components" [Eisenberg, 2001]. We and others [Kocoman et al, 2001] believe that these components can be dynamically bound using completely automated processes. Furthermore, these processes can be tailored around a user using techniques developed in the areas of adaptive hypermedia [Conlan et al, 2003].

In this paper, we will present how we automatically create compositions using A.I. planners, how these compositions are then analysed for weaknesses and how we then recreate sections of the plans to provide the user with a more

personalised experience. The findings of our experimentation are presented and we then discuss the research that needs to be carried out before we can achieve our vision of how adaptive, personalised service composition can take place.

2 Motivations

A Web service, that is a software component that uses technologies originally found on the World-Wide-Web for communication and self-description [Scott, 2001], provides an open and lightweight means with which users and other pieces of software can interact. Chaining these services together by feeding the outputs of one into the inputs of one or more other Web services is a simplistic but fundamentally correct view of what Web service composition is. Because the dynamic linking of these software components is so easy to achieve (purely a run-time process), with a sufficiently large volume of services available, the generation of tailored software which at least corresponds to the functional requirements of a user or entity is entirely feasible. When we have a richer set of descriptive mechanisms both on the non-functional properties of the service and the requirements and preferences of the user, it becomes possible to customise the composition very specifically to the user [Higel et al, 2003]. This adaptivity opens an array of possibilities in many established and emerging areas of computing. We believe that the two most significant ones are those of ubiquitous computing and eBusiness.

In the domain of ubiquitous computing, where intermittent and location dependent availability of services becomes a factor, a more flexible approach to user interaction with the environment and services is desirable. Web services, with their openly defined (and arbitrarily verbose) interfaces provide a suitable encapsulation of environmental functionality [Issarny et al, 2005] and through the support of ontologies offer ample means of abstracting Web services so that they can be resolved to instances providing the correct functionality at run-time. As users begin to require more complex interactions with their surroundings, service composition can be used to create and manage these requirements.

As various high-profile organisations begin to provide some of their business functionality via Web services (specifically SOAP and WSDL)[1], it is our belief that through appropriate semantic support, the next generation of online commercial interactions may be entirely facilitated by Web service composition. Businesses which realise their core competencies can focus purely on providing the parts of a business interaction at which they excel and allow others to do the same with other parts of said interaction.

In both of these research areas, an improved user experience will lead to improved user acceptance. We believe that transparency and adaptivity are a key to driving this. To facilitate this these, we require a rich means of expressing the needs of a user and any other requirements that might be imposed by their environment or any entity that they are representing.

[1] See http://soap.amazon.com/schemas2/AmazonWebServices.wsdl and http://www.google.com/apis/ for two prominent examples.

3 Existing Research and Technologies

3.1 Automatic Service Composition

Automatic Service Composition, the generation of chains of services by software, has largely grown from the application of established knowledge engineering and work-flow management techniques to those of artificial intelligence. Service composition based on incremental planning, hierarchical task network planning and graph theory all achieve their tasks effectively with varying degrees of automation. Languages used to describe service compositions like BPEL4WS and WSCL have evolved from work-flow description languages.[van der Aalst et al, 2003]. As previously mentioned, the use of adaptive hypermedia techniques can do much to guide research into the tailoring service compositions to individual users.

What we are presently lacking is a general mechanism for describing how to aggregate and reason on the non-functional properties of services and how to analyse the suitability of a chain of services when compared to a user's needs and requirements. Work has been done to solve specific portions of this problem, for instance [Jaeger et al, 2004] goes into some depth on how to model and aggregate different sub-concepts in quality of service while [Quinn et al, 2005] explores the same idea in the realm of trust and security. It should however be clear that a general mechanism for expressing arbitrary aggregations would be preferable, rather than relying on specific and differing rule-sets for each class of non-functional property.

3.2 A.I. Planning

Artificial Intelligence Planning creates sequences of pre-defined actions to alter a given set of environmental entities from one supplied set of states to another desired set of states. The description of these actions consists of a set of required states (for instance a book-selling action might require that a given person actually wants to buy a book), a set of resultant states (using the previous example, the action might result in the required book being ready to ship to a user).

The Problem Domain Definition Language (currently at version 2.2) is used to describe two general concepts to a planner[2]. The first is the domain description, which outlines the available actions, their preconditions and outputs and other configuration options that the planner should use. The second is the problem specification, which contains a list of all of the instances of objects that are to be considered when creating the plan. More importantly, the problem specification supplies the planner with a list of current environmental states and a list of desired environmental states (see Fig. 1.). Upon being executed, the planner will search through the available actions for a path which leaves the environment in the desired state, iteratively improving the plans until either the search space is exhausted or it exceeds the alotted time.

[2] See http://users.raise.anu.edu.au/~thiebaux/workshops/ICAPS03/ for a number of publications from the ICAPS Workshop on PDDL, 2003

```
(:init (haspaymentconfirmation companyd moneyamounte personb ) (readyto-
ship book
titlec personb ))
(:goal(hasbook booktitlec personb ))
```

Fig. 1. PDDL representation of a user's initial state and goals

An evaluluation of the planners which competed in the 2004 International Planning Competition eventually brought us to use LPG-td [Gerivini et al, 2004], which has full support for the two new requirements for PDDL2.2 compliance (derived predicates and timed initial literals, neither of which had any particular bearing on our research). In terms of flexibility, it allowed us to very quickly experiment with different search algorithms, maximum search times and search depth.

4 Current Work

We have developed an API for generating PDDL, analysing the output of the planner and affecting the "rankings" of the available services to reflect user preferences. In this section, the architecture and processes involved in using these pieces of software is discussed.

4.1 Architectural Overview

From a component-oriented perspective, our architecture at present consists of a service repository, a PDDL generator, a planner, a planner output analyser and a replanner-controller. The service repository is a relational database which stores the functional and non-functional properties of the available services. It is our intention to replace this with an XML database which will store OWL-S documents, as certain properties of relational databases make this type of work somewhat cumbersome. There is a lack of openly available, composable service descriptions, so we have developed a set of our own service descriptions using the by now customary "buying a book" example. Of note is our treatment of the concept of a book. We treat an electronic representation of a book as being equivilant to that of a "dead-tree" copy of the book.

4.2 PDDL Generation and Basic Adaptivity

It should be quite apparent that there is no one-to-one mapping between each concept in Service Composition and A.I. planning. Planning deals purely with the functional and practical requirements for and outcomes of executing an action. Research into service composition has developed a richer set of descriptive mechanisms for expressing properties beyond those which are functional. What mapping can be done is relatively straightforward (though the same cannot be said for the actual generation of syntactically correct PDDL!).

We use PDDL's durative actions to represent each service from our repository in the PDDL domain representation for reasons that shall be outlined later in this section. The list of types and predicates, required as part of the domain representation are all derived and sanity-checked from the available actions provided by each service. The PDDL problem representation generator is fed a list of predicates and goals required by the user which are converted into an appropriate form (see Fig. 2.).

```
(:predicates
(wantsbookgenre ?a - bookgenre ?b - person)
(wantsbook ?a - booktitle ?b - person)
(owesmoney ?a - company ?b - moneyamount ?c - person)
(haspaymentconfirmation ?a - company ?b - moneyamount ?c - person)
(readytoship ?a - booktitle ?b - person)
)
```

Fig. 2. PDDL representation of predicates, required as part of the domain representation

PDDL durative-actions differ from other PDDL action representations by allowing the PDDL author to specify the duration of the action's execution time (see Fig. 3.). Because a planner, when presented with two actions of differing durations but identical functional properties, will prefer that with the lower duration, we are presented with an opportunity to perform limited adaptivity. In essence, part of our experimentation involves the distillation into a single number of the user's preferences and any other relevant policies applied to the available services, allowing the planner to create an adapted and optimal plan.

```
(:durative-action fastbookrecommendation
:parameters (?a - bookgenre ?b - person ?c - booktitle )
:duration (= ?duration 71)
:condition (at start(wantsbookgenre ?a ?b ))
:effect (at end(wantsbook ?c ?b ))
)
```

Fig. 3. PDDL representation of a book recommendation service

4.3 Service Classification

A reduction of the search space in any planning problem will decrease the execution time of the planner. The abstraction of service functionality back to classifications based on their functional properties allows us to delay committing to using a given service until near-execution time of the composition. These two

observations are what motivates our use of a service classifier. This analyses a set of services and groups them by like functional properties. As an aside, it should be noted that in a real-world environment, this would require some ontology mapping support to bridge gaps in the terminology used by two different service providers. Instead of feeding "raw" service descriptions into the planner, we instead feed classifications which can be then resolved at a later date, based on both service availability and user requirements.

4.4 Instantiating the planner, Interpretting its output, Performing Adaptivity

The planner is fed a problem file, a domain definition and various parameters governing the nature of its execution (search type, maximum execution time, search depth, verbosity etc.). We simply fork the process and wait for it to complete its execution. The planner prints out a list of all of the solutions it has produced, along with the files in which it has stored these (see Fig. 4.). The output is parsed and the string representations of the service classes or services are resolved back into the internal representations of those service classes within the software.

The service classifications must then be resolved back into appropriate services, based on supplied user requirements. At present, we have a simplistic, normalised representation of user non-functional properties which are placed into a matrix and multiplied by a similar matrix derived from the available services. An average is generated from the elements of the resultant matrix which is then used as a suitability metric for this service. The highest scoring service is selected and placed into the appropriate point in the solution. This simplistic representation of user preferences and service non-functional properties is a stop-gap, whose replacement is outlined in the *Further Work* section.

```
0.0003: (CHEAPBOOKRECOMMENDATION BOOKGENREA PERSONB BOOKTITLEC) [76.0000]
76.0005: (FASTBOOKSELLING BOOKTITLEC PERSONB COMPANYD MONEYAMOUNTE)
[78.0000]
154.0007: (FANTASTICPAY COMPANYD MONEYAMOUNTE PERSONB BOOKTITLEC)
[82.0000]
236.0010: (WONDERDELIVERY COMPANYD MONEYAMOUNTE PERSONB BOOKTITLEC)
[75.0000]
```

Fig. 4. An example of a solution file generated by the planner

4.5 Replanning

If the resolution of a service classification back to an existing service provides a poor match, it might be possible to generate a plan which is longer (i.e. consists of more services) but which in a non-functional sense, is more closely matched

with what the user desires. A user might prefer to have a PDF emailed to them rather than a "dead-tree" copy sent through the post, even if the composition involves invoking a larger number of services. Assuming our descriptions of non-functional properties can expose the shortcomings of the shorter solution, our software is capable of removing the offending classification from its list of services and can then attempt to generate a new plan to bridge this gap in our overall service composition.

In our experimentation, we created two possible paths through a given composition. A book could effectively be purchased in "dead-tree" format and shipped as a parcel through the postal service or as an eBook that would be converted into a PDF and then emailed to the user. By modifying the user's preferences, our planner-output interpretter would identify a potential weakpoint in the plan and then instruct the planner to attempt to generate a new solution for that portion of the composition.

4.6 Experimentation

Given that we want to find out how best to adapt based on the information expressed in the services' non-functional properties and the user's view thereof, we have attempted to base our experimentation around the following questions:

- Is the classification of services before the planning process and then replanning the weakpoints the most logical technique for maintaining optimal adaptivity (see Fig. 5.)? Or can we leave all adaptivity down to the planner using the above techniques (see Fig. 6.)? This was examined by running the composition process repeatedly and measuring its execution time and comparing the suitability of the composition to the user each time. This appropriateness is examined by comparing each constituent service to its functionally equivilant peers and examining if it is the one best suited to the user. Because we lack a proper means of aggregating and comparing chains of services of different lengths, for now, this is the only test we can perform.
- Can the modification of a user's preferences completely alter the resultant plan, for instance resulting in a plan which emails a PDF version of the requested book to a user rather than shipping a copy of the book through the post? This could be termed taking a different *functional path* through the solution space.

The experiment was carried out on an 800Mhz Pentium 3 running Linux. The wrapper and APIs around the planner were written in Python. Our pool of initial services consisted of 62 distinct services, which could be classified into 7 different classes. The experiment was run 1000 times for each approach.

5 Results

5.1 Classification vs. Pre-rating

Our experimentation with the pre-rating technique showed that over 1000 executions, the average time to generate the composition was 1.969 seconds and

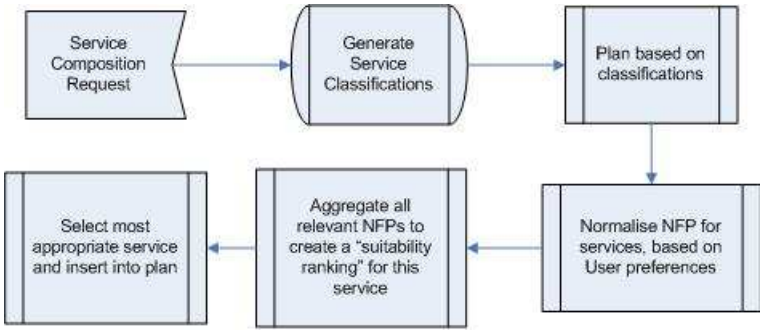

Fig. 5. Service Composition using pre-classification and post-analysis

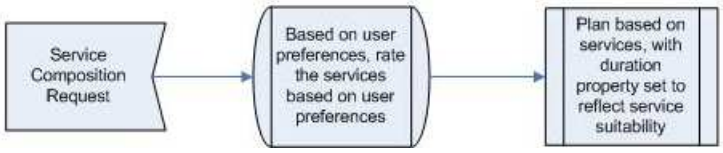

Fig. 6. Service Composition using pre-rating of services

had a 100% match rate to a composition pre-determined to have been the closest match to a user's preferences, as best as this can be defined at this point. Classification and post-planning resolution, over the same number of executions, averaged 2.103 seconds and again, had a 100% accuracy rate. The difference between the two is barely discernable, so it stands to reason that we can evaluate both techniques on what we believe to be important: their conduciveness to user-centric-adaptivity.

5.2 Plan Modification

We modified the supplied user preferences to quite blatantly express their desire for low-cost solutions at the expense of reliability, security and latency. The functional path which resulted in a PDF version of the book being emailed was set to be practically free. In this case, the planner did select plans based on this path as being the most suitable option. When using the classification and replanning approach, the replanner spotted a potential weak-point in the plan and replaced it with the same PDF generating composition.

However, it is simple to represent cost as a durative property that the planner understands. Non-functional properties which cannot simply be aggregated by adding them together cannot be treated in the same way, so relying on the planner to perform this adaptivity will never provide a general solution. Even if we intelligently distill all non-functional properties into a single suitability metric, without severely skewing the *duration* properties of the actions, which will require hand-tuning on a case by case basis, the planner will never have

sufficient understanding of the user's preferences such that it might generated a longer but more suitable plan. For this reason, it is only through analysis of weakpoints within the plans and replanning based on the functional properties of that point that we can adapt and refine the compositions on a more "informed" level.

It is this exact idea that we intend to explore further.

6 Research Directions and Further Work

It became quite clear throughout the implementation of this experiment that simple normalised representations of a user's view of the importance of a given non-functional property or the service's description of its own non-functional properties are insufficient. In reality, we require an appropriate means of describing how to manipulate and reason on non-functional properties for the adaptivity and flexibility to be useable in a real-world scenario. We have identified three primary types of non-functional property aggregation:

- When two different users refers to a non-functional property like "cost", they might imply different things. For one, the initial outlay might be extremely relevant, for the other, the total cost of ownership would play a larger part in this definition. If the Web service defines both of these, we should allow the user to specify a ruleset for aggregating seemingly irrelevant NFPs for the service into ones that are meaningful to them. Once this is done, the planner and analysis software can properly make decisions on how to compare two given services.
- A service could be described in terms of its cost, reliability or security. When aggregating these together to form a single suitability metric for a service, we may find that some non-functional properties conflict or need to be aggregated in unique ways. A non-functional property might be represented non-numerically (a service might only be available on "Wednesday") and we intend on providing a means of expressing the relevance (or lack there of) of such properties on the overall suitability of a service.
- When attempting to analyse the effectiveness of a chain of services based on user preferences, we are presented with the simple reality that aggregating like NFPs across multiple services requires rulesets defining how this should occur. To aggregate the cost of using a set of services is trivial: we add the cost of each service and compare it to another chain to find the cheapest. However, when analysing the security of such a chain, we might take the value of the weakest link. The mean time between failures might be calculated as an average of all of the services being used. For this reason, a descriptive mechanism is required to aid these decisions.

7 Conclusions

In this paper, we have outlined the techniques we have used to generate trivial service compositions with limited adaptivity using an incremental planner. We

have demonstrated why we feel pre-composition service classification is conducive to increasing the adaptivity of the composition itself. Also, we have deduced that replanning is key to creating refined and personalised compositions, and that having a suitable means for describing appropriate aggregation techniques for the non-functional properties of services is critical to adapting service compositions to a user's specifications.

References

[Eisenberg, 2001] "Preparing for the Web Services Paradigm", Eisenberg, B., W3C workshop on Web Services, 2001

[Kocoman et al, 2001] "Towards Zero-Code eService Composition", Kõcõman E., Melloul L., Fox A., Hot Topics in Operating Systems, 2001

[Conlan et al, 2003] "Applying Adaptive Hypermedia Techniques to Semantic Web service Composition", Conlan O., Lewis D., Higel S., O'Sullivan D., Wade V., Adaptive Hypermedia 2003

[Scott, 2001] "The Road to Web Services", Scott, G., W3C workshop on Web Services, 2001

[Higel et al, 2003] "Towards an Intuitive Interface for Tailored Service Compositions", Higel, S., O'Donnell, T., Lewis, D., Wade, V., 4th IFIP International Conference on Distributed Applications & Interoperable Systems, 2003

[van der Aalst et al, 2003] "Web Service Composition Languages: Old Wine in New Bottles?", van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M., 29th Euromicro Conference, 2003

[Jaeger et al, 2004] "QoS Aggregation for Web Service Composition using Workflow Patterns", Jaeger M. C., Rojec-Goldmann, G., Muhl. G., edoc, vol. 00, no. , pp. 149-159, Enterprise 2004.

[Quinn et al, 2005] "deepTrust Management Application for Discovery, Selection, and Composition of Trustworthy Services", Quinn, K., O'Sullivan, D., Lewis, D., Wade, V.P., 9th IFIP/IEEE International Symposium on Integrated Network Management (IM 2005), 2005

[Issarny et al, 2005] "Developing Ambient Intelligence Systems: A Solution based on Web Services", Issarny, V., Sacchetti, D., Tartanoglu, F., Sailhan F., Chibout, R., Levy, N., Talamona, A., Automated Software Engineering, Volume 12, Issue 1, 2005

[Gerivini et al, 2004] "LPG-TD: a Fully Automated Planner for PDDL2.2 Domains", Gerevini, A., Saetti, A., Serina, I., International Planning Competition, 14th Int. Conference on Automated Planning and Scheduling (ICAPS-04), 2004.