Ph.D. Thesis Extended Abstract: Planning for Web Services

Mark James Carman
Univesità degli Studi di Trento
carman@irst.itc.it

Abstract

In this thesis I propose to develop a system capable of performing automated planning and execution of web service operations in order to achieve user defined goals. Web services are services described in XML and made available over the internet. The proposed system will use the information available in "standard" service descriptions and will not rely on formal mark-up of such descriptions. I have performed some preliminary work on XML Schema type matching and an algorithm for automated service composition, which when combined allow the user to perform intelligent information retrieval. In this thesis I intend to extend on these techniques to handle world-altering goals. I plan to investigate service discovery based on meta-data descriptions and the matching of service state-machine descriptions. I then plan to incorporate a learning framework capable of teaching the planner to recognise the semantics of certain operations within the domain based on appropriate example documents.

Introduction

The wealth of information available on the internet is currently being complemented by an ever-increasing number of services. These services offer the possibility not only to gain more specific types of information but also to interact with the sources of the information, changing the state of these systems and causing real world processes to occur. Our aim is to plan for the automated discovery, composition and execution of such services in order to achieve user specified goals. The ability to perform automated service composition would revolutionise a number of application areas including e-commerce and systems integration. For example in online travel, a planning system capable of discovering and interacting with flight and accommodation booking services could automatically arrange business trips based on user preferences.

We are building a framework for automated service composition based on the information available in service directories and interface descriptions. In our framework we do not rely on the existence of semantic mark-up in service descriptions (as advocated by the Semantic Web), but take a client side approach to the semantic interpretation of such descriptions. At present the framework consists of a semantic type matching algorithm and a planning and execution algorithm. We envisage extending the system to handle service matching and to learn to recognise the semantics of documents from examples.

Background

Web Services facilitate platform-independent application integration between enterprises. Other integration technologies (such as CORBA or Java RMI) require companies to standardise on a common platform. Instead, by using XML Schema to format data in a self-describing and programming-language independent manner, and SOAP to encapsulate and send the data over a transport (e.g. HTTP), companies can link applications without needing to standardise their platforms. Using the Web Services Description Language (WSDL) they can describe the service interfaces they provide and publish these descriptions in a service registry, so that other companies can discover and use them.

The Problem: Planning for Web Services

Services are made up of operations, which can be seen as actions that a planning system can perform on the world. Unfortunately, from a planning point of view, these actions are not fully specified: their real world effects are unknown, as are the preconditions on their input variables, and the conditional effects on their output variables. The problem of planning for web services is further complicated by the fact that the information describing the domain is distributed (infinite), heterogeneous and incomplete. Heterogeneity is a problem both in terms of the data that services input and output, and the descriptions of the services themselves. The problem of data heterogeneity is due to the fact that there is no common high-level data model between services. In some cases service providers will adopt a common set of industry specific datatypes. In most cases however, an amount of heterogeneity in data descriptions will remain. The problem of service heterogeneity has to do with the fact that two logically equivalent actions may be called differently in different services and require different patterns of interaction prior to their execution. For example one might need to perform a "login" action followed by a "getCatalogue" action in one service before executing "purchaseItem", while in another service the logically equivalent action "submitPurchaseOrder" can be executed without executing any prior login procedure. In this thesis I propose to address both heterogeneity problems.

Different Approaches

One approach to solve these problems is to explicitly describe the semantics of service operations in the interface documentation. In this case the service provider would annotate each service interface document with "real world" preconditions and effects, (described using semantic web standards such as OWL and DAML-S). For example, she could state that the effect of the "buyBook" operation is to make the $own(book)$ predicate true. Then a planner, which has the goal of arriving in a state in which this predicate holds, can conclude that it needs to execute the operation "buyBook" by giving it the relevant inputs. Note that, if the ontology (definition of own) used to describe the service is not the same as that used by the planner, then a matching of these ontologies must also be created. Manually creating such mappings is a non-trivial task, while automated mapping between different ontologies is very much still an area for future research.

In my work I do not assume that semantic mark-up is present in the service descriptions, because in the short term at least, there is little economic motivation for the providers of web services to create such mark-up. Semantic descriptions would make sense in a closed environment (such as a particular vertical industry), where all parties can agree on a common ontology. There has been some previous work on planning for web services based on semantic service descriptions. In (McIlraith & Son 2002), the authors investigate the problem of instantiating different precompiled plans based on user preferences, while in (McDermott 2002) the planning domain description language PDDL, is extended to handle information producing actions.

A different approach to automated service composition was presented in (Thakkar *et al.* 2002), where services are modeled as web information sources for which a common data model is already known. A common data model means that database query planning and transformation techniques can be used for plan synthesis and optimisation. The authors envisage a scenario, in which automated wrapper generating software is used to create standard service interfaces to information provided on different web sites. Thus the problem of data heterogeneity is left to the wrapper software, and the standard service interface means that service heterogeneity is also not a problem.

Our Approach

Our approach to planning for web services is a pragmatic one based on the information that is currently available in service interface definitions. We do not require that service providers describe their interfaces using semantic mark-up, nor that they limit themselves to the use of "a standard data-model". Instead we attempt to perform planning based solely on the information that is already available in the service descriptions:

1. operation input/output data signatures
2. service process descriptions
3. service description meta-data (business taxonomies)

The set of operations provided by a service can be found in its WSDL description. The i/o signature of each operation tells us what type of document needs be provided in order to execute it, as well as the types of documents that will be returned upon successful and unsuccessful execution. The signature also gives us information on possible compositions of services. For example, if a particular service has an operation "buyBook", which takes as input an "isdnCode", and another operation "getISDN" (from a different service) outputs values of the same type, then the planner may try to execute the latter in order to generate input for the former.

The i/o characteristics of operations in a service give only a static description of the functionality of the service. One can describe the fact that "selectItem" and "purchaseItem" are both operations provided by the service, but not the restriction that the former should proceed the latter during execution. Service process definitions (described using emerging choreography standards such as WSCI and BPEL4WS) can provide such procedural information to the planner.

Service classification information is found in UDDI service registries, in which services are classified according to industry-segment, provider, location, etc. If standard classification indices are used to describe the services, then the meta-data becomes useful to the planner when searching for relevant services and for inferring similarity between them.

According to our alternative approach, we do not have any description of the real-world effects of service operations. Thus we need some way of expressing goals in the system that is independent of these effects. We can do so by describing goals as information requirements - as a type of document that the planning system needs to create. By placing restrictions on the values of fields within the requested document, one can express a goal such as $own(book)$ as: create a "purchaseOrderConfirmation" document within which the value of the "purchaseItem" field is the name of the desired book. In order to satisfy the goal, the system needs to provide a document (an output from a service), which is of similar type to the goal and adheres to the given constraints. These constraints could be equalities such as "bookName" equals "Harry Potter and the Philosopher's Stone" or numerical inequalities such as that the "price" field has value "< 20 Euro".

A planning problem is then a combination of a goal with some "local information", which is available to the planner to use as input when executing services. An example of such local information could be the name, address, and credit card details of the person requesting the goal. The local data together with the goal value restrictions can be seen in some way as defining the initial state of the planner.

Progress to Date

In this section I outline briefly the work I have been doing thus far toward the goal of automated planning for web services. This work is important in so far as it provides the basis for the research work proposed in the next section. The work is divided into two parts: work on a type matching algorithm capable of discovering semantic equivalence between similar datatypes, and a service composition algorithm for use in composing services to achieve information goals.

Type Matching

In order to compose services from their operation definitions, we need to be able in some way to match different datatypes, such as the goal with various service outputs. The ability to discover matches between identical types is not sufficient for our purposes as we cannot guarantee (indeed it almost certainly not the case) that the required services will input and output types from a common schema. Thus we need to tackle the problem of data heterogeneity, which is to decide if under some mapping the data described by one datatype can be substituted for that described by another. I.e. if we can take the output produced by one service, map it, and use it as input for another service.

A datatype has a set of values that can be considered as representing different possible states of the world. For example when receiving a message of type "Person" with the field "name" equal to "Peter" and "age" equal to "21", we can interpret the message as saying that there exists a person called Peter who is 21 years of age. And if another message arrives, this time of type "UniversityStudent" with name and age as before, we can interpret it as saying that there exists a person Peter who's 21 and goes to university. The second message describes a smaller set of possible worlds (interpretations) than the first, i.e. the cases where Peter is not a school student nor a worker, but a university student. Now if we needed to fulfill a goal (or provide an input to a service) of type "Person", then an instance of the message "UniversityStudent" can be used to provide the required information. If however we require an input of type "UniversityStudent" and have a message of type "Person" the reverse is not possible as we do not know whether or not the instance to which the message refers is a university student or a school student, and so on. Thus in order to be able to map from one datatype to another we require that the latter describes a superset of the possible worlds that can be described by the former. I.e. that the target type is a more generic version of the source (under a given mapping).

Now in our algorithm, when we compare the goal type t_{goal}, to a particular service output type t_{out}, we require that $t_{goal} \supseteq_M t_{out}$, which is to say that all documents conforming to the output type also conform (form a subset of those conforming) to the goal type after a certain mapping M has been applied to them. For example, a goal such as:

```
<Weather>
 <Temperature type="decimal"/>
 <Location type="string"/>
</Weather>
```

with a restriction that value of field "Location" should be "Adelaide", should match against a schema such as:

```
<DailyWeather>
 <LocalConditions>
  <AmbientTemperature type="decimal"/>
  <Rainfall type="decimal"/>
 </LocalConditions>
 <Address>
  <City type="CityNames"/>
  <State type="StateNames"/>
 </Address>
</DailyWeather>
```

where:

```
<simpleType name="CityNames">
 <restriction base="string">
  <enumeration value="Adelaide"/>
  <enumeration value="Melbourne"/>
  <enumeration value="Perth"/>
  ....
 </restriction>
</simpleType>
```

This is because the information required by the first can be found from within the second. I.e. the values for "AmbientTemperature" and "City" in the second can be mapped to "Temperature" and "Location" in the first. Note also the fact that the value "Adelaide" (which is a restriction on the field "Location" in the goal), is one of the possible values of the type "CityNames" in the output type. I.e. some instances of the output type adhere to the value restrictions in the goal.

We use WordNet (Fellbaum 1998) as a lexical resource for performing matching of labels within the type structures. We first use it to match synonyms like "car" and "automobile", as most documents which refer to an instance of car are also referring to an instance of automobile. We then try to take advantage the other relations which exist in WordNet. One can do so in a qualitative or quantitative manner, we take the qualitative approach, using the WordNet noun hierarchy to find generalisation relationships such as "car"$\supseteq$"vehicle" (car is a type of vehicle) or "city"$\supseteq$"location". As part of my thesis work thus far, I have developed an algorithm which uses these relations to decide whether one schema type can be seen as a generalisation of another under a given mapping. Details of the algorithm can be found in (Carman, Serafini, & Traverso 2003).

Service Composition Algorithm

Having discussed an algorithm capable of matching and mapping data between heterogeneous type structures, we outline an algorithm that exploits this capability to compose and execute service operations to retrieve desired information. Whenever we execute an operation within a service we cannot guarantee that it will execute properly, providing the desired output (e.g. where "location" equals "Adelaide"). So we are forced to interleave search and execution in order to overcome this problem of incomplete knowledge regarding the domain.

The algorithm takes as input the goal to be achieved and searches a UDDI directory for all services which are capable of outputting documents of sufficient similarity to the goal, using the type matching algorithm described previously. The service interface with the most similar output is selected first. If there is more than one implementation of that interface, the algorithm will select one of them based on meta-data values. It then attempts to execute the particular service operation that produces the desired output. Before doing so, it must create the required input document. It starts by using the immediately available information, such as that given in the goal, the local information, and past input and output documents if they exist. If the available in-

formation is not sufficient, the algorithm must again search the outputs of other services, i.e. the procedure calls itself recursively. Generally, not all of the data required to fill the input document will be contained in a single source, thus the process repeats on sub-elements of the input document until a complete document is produced or a search limit is exceeded. Having generated an input document, the algorithm attempts to invoke the operation. If it does not produce the desired output, the algorithm rolls back certain decisions made when creating the input and tries again. The heuristic guiding this search can be based on the confidence the algorithm had in its decision at each point, i.e. the quality of the match. If after a "reasonable number" of attempts, the operation still can't be executed, then the problem may be the data given as input to the previous (successfully completed) operation. Thus the system either tries to re-execute the previous operation with different inputs, or gives up on the service altogether and searches for a new way of achieving the goal.

The search tree created by the above algorithm can be seen as an AND-OR tree, where the "OR" branches represent different ways of creating an input, and the "AND" branches represent combinations of service outputs that together produce an input. Leaves in the tree represent data found to be available locally. The execution algorithm described above performs a bounded best-first search through the tree, where the bound sets a limit on the number of failed execution attempts allowed for completing a given sub-tree. The execution bound is decremented for each level of decent in the tree.

This algorithm assumes that all of the operations within each service are atomic, and that the service to which they belong is stateless. I.e. there are no ordering constraints on the executions of operations within a given service. In some cases this assumption may be false, and the exact ordering of operations may be critical for the correct execution of services. For example a service might require that a "login" operation is performed prior to executing a "getStockQuote" operation. The algorithm described above would never try to invoke the former operation and thus would never be able to successfully execute the latter. In some cases such information may be available however in the form of service process descriptions. Such process descriptions may even provide additional information regarding the flow of data between operations within a service (i.e. fields in input and output documents that refer to the same value). In (Carman & Serafini 2003), we describe a more complicated algorithm that takes service process descriptions into account when executing services.

Research Plan

I propose four areas of work in this Ph.D. thesis. The first two areas involve extending and improving the type matching and execution algorithms described previously. My goal in this case is to produce a working system for intelligent information retrieval. The third and fourth areas involve extending the general framework with enhanced service discovery and matching capabilities, and with a learning framework for recognising the semantics of service operations automatically. The aim of the latter two work areas is to produce a planning system capable of handling "world altering" goals such as to "buy a particular book".

Type Matching

There is a large amount of information available in WordNet, and by using the noun hierarchy we have only touched the surface of what is possible. I intend to investigate the use of other relationships found in WordNet, such as the meronym (part of) and the antonym relations. For example, if it were the case that "stock symbol" and "company" were related though an inherited "part of" relation, then one could perform the match "company" $\supseteq$ "stock symbol" by inferring that an instance of the concept "company" must exist in the second type in order that an instance of one of its parts can be referred to. (This is equivalent to introducing a "company" node above "stock symbol" in the more specific type). Use of this "part of" relation is complicated, however, by the problem of mapping between the data values, restricting its use to cases where a service capable of performing such mapping has been discovered.

I plan to investigate the use of statistical techniques (based on relative frequency of words and senses) in the matching algorithm. For example, it would make sense to place more importance on the matching of less-common words, like "hydroflurocarbon", than more common ones like "chemistry". By doing so one could improve the type matching algorithm by making it more robust against miss-matches due to words which carry multiple senses.

A third area for improvement would be to extend the type matching algorithm to take instance-level data into account, when discovering matches. For example, it would make sense to use the fact that the instance value "Adelaide" is also found in WordNet and is found to be a type of "city" thus making the substitution of "city" for "location" all the more plausible. This type of instance level reasoning / consistency-checking could be very useful during service composition when partially instantiated documents become available.

Service Composition Algorithm

As regards the service composition algorithm, I plan to investigate more complex search patterns to improve the performance and reliability of the system. Techniques such as "limited discrepancy search" could be used to improve backtracking performance. We could also investigate the use of "wider" search patterns with the aim of minimising the number of steps (operation invocations) in the information gathering plans before execution is attempted. The intuition here, is that a plan with fewer steps is more likely to execute properly with less modifications. Furthermore, since we are performing information gathering, we could attempt to execute a number of different plans in parallel. Parallel execution makes sense in an internet environment where the time needed to execute individual service operations is large.

At present the planning system operates without any semantic "understanding" of the effects of operator execution. We could encode the goal "buy book" using a certain "purchase order confirmation" document schema, by restricting

values on certain fields in the document, but the trial and error execution system might end up buying 100 books before it gets the right one! It would do so, because it has no knowledge of the "real world" effects of its actions. A partial solution to this problem would be to make use of information regarding the transactional behaviour of a service, if it is available in the service process description. If a service states that the execution of a particular operation can be undone, then the planner knows it is safe to continue with its execution. In the case where an undesired output is produced (i.e. the planner purchases the wrong book), the planner can simply roll-back the execution of that operation and try again. Thus in order to prevent unintended purchases, it would be sufficient to limit the amount of money available to the planner. Such a spending limit could be described without the use of a domain ontology by placing restrictions on values of fields in the set of output documents created during plan execution.

Service Discovery and Matching

I intend to investigate taxonomy matching algorithms for use during service discovery. The ability to find equivalent instance values from different classification structures (such as UN-SPSC and NAICS codes) could be useful both for discovering matching meta-data service descriptions and for discovering matches between input and output document types. For example a particular company may be classified as "software and hardware vendor" under one scheme and "software supplier" under another, in which case the planner should know that the two classifications can be considered equivalent.

Secondly, to handle the semantic problem described in the previous section, I intend to enable the planner with a local partial model of the domain (described in terms of a simple local ontology) and use a technique similar to type matching but of services process descriptions to handle the service heterogeneity problem. The idea here would be to find matching operations in process diagrams describing two different services, where for one of the services some of the "semantics" of operations is already known (in terms of formal preconditions and effects). In order to find such matching operations their names, input and output datatypes, and positions within process graphs would be compared. Once a matching operation has been found, the semantic description of the original operation could in some way be "inherited" by the second. The details of such an inheritance need to be investigated. This idea of "schema matching" can be seen as a generalisation of the idea of replace-ability of web services described in (Mecella, Pernici, & Craca 2001), where the authors find exact equivalence between services by performing bisimulation and matching (identical) labels.

Learning from Examples

Thus far in our attempt to perform planning for web services we have used the information available in the service descriptions and have augmented that with information available in the lexical resource WordNet. Another source of information which is available a-priori to the planner is the ever increasing set of standards for describing e-commerce interactions between companies. There are a number of standard protocols (such as UBL, xCBL, cXML, etc.), which describe a standard set of documents that must be sent between trading partners during a business exchange. One idea would be to prime the planner with some knowledge (formal semantics) regarding these standards, so that if it does happen to interact with a service described using a standard it can take advantage of that information. More importantly, however, may be the possibility of generalising some of this knowledge, so that when the planner interacts with a service described using labels that are similar to those used in one of the standards, it may automatically draw some conclusions about the meaning of the interface being presented.

One way to do this would be to find common parts to the documents described by the different standards and attach a common semantic to these sub-structures in terms of a local ontology. One could use the type matching algorithm described above to discover semantically equivalent structures in different documents, (such as the "address" field in a purchase order document). Classifiers could then be trained to recognise certain types of documents based on the substructures they contain. I.e. the system could distinguish between a "login form", a "purchase order" and a "purchase confirmation" document. It could then conclude which operation performs the "purchase action", and only execute that operation if it is sure that the inputs are correct. I.e. we could train a classifier to recognise important operations for which there is a clear semantics.

References

Carman, M., and Serafini, L. 2003. Planning for web services the hard way. In *Workshop on Service Oriented Computing, International Symposium on Applications and the Internet (SAINT-2003)*. IEEE Computer Society Press.

Carman, M.; Serafini, L.; and Traverso, P. 2003. Web service composition as planning. In *Workshop on Planning for Web Services, 13th International Conference on Automated Planning and Scheduling (ICAPS 2003)*.

Fellbaum, C., ed. 1998. *WordNet: An Electronic Lexical Database*. The MIT Press.

McDermott, D. 2002. Estimated-regression planning for interactions with web services. In *AI Planning Systems Conference*.

McIlraith, S., and Son, T. 2002. Adapting golog for composition of semantic web services. In *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002)*. Morgan Kaufmann.

Mecella, M.; Pernici, B.; and Craca, P. 2001. Compatibility of e-services in a cooperative multi-platform environment. In *2nd VLDB Workshop on Technologies for e-Services (VLDB-TES 2001)*. Springer.

Thakkar, S.; Knoblock, C. A.; Ambite, J. L.; and Shahabi, C. 2002. Dynamically composing web services from online sources. In *Workshop on Intelligent Service Integration, The Eighteenth National Conference on Artificial Intelligence (AAAI)*.