AI Planning for Automating Web Service Composition in Tourism Domain

Husniza Husni

This report is submitted as partial fulfilment of the requirements for the Honours Programme of the School of Computer Science and Software Engineering, The University of Western Australia, 2005
Abstract

Web services are changing the way how online business operates, especially in tourism domain. Typically, existing Web services are built individually as atomic services. The rapid growth of Web services has created the need for Web service composition so that clients can compose atomic services to achieve more complex tasks. Thus, to ease the process, automation is important. Automation means that the service composition is done with less or no user interference. Hence, we propose a framework to automatically compose Web services using SHOP2 planner. SHOP2 is a planner that implements AI planning technique, called Hierarchical Task Network (HTN). We propose and implement a framework to compose services available from the Australian Tourism Data Warehouse (ATDW) and present the example execution results. We also outline some drawbacks of our approach, identify open problems, and suggest future work to improve the framework.

Keywords: Web service composition, automatic composition, AI planning, SHOP2, ATDW

CR Categories: D.1.3, D.1.6, D.1.5, I.2.8
Acknowledgements

Thank you God for all the blessings.

I would like to thank Dr Wei Liu for her continuous support and guidance during the research period. Without her help, this research will not have been materialized. Thanks also to the Australian Tourism Data Warehouse (ATDW) for the permission to access and consume the services provided. I would also like to thank my family and Suhairizam Omar for their love and endless motivational and emotional support throughout the year. Finally, thank you also to my friends for all the help and encouragement.

In memory of Jimmy, Timmy, and Abu.
# Contents

Abstract ii  
Acknowledgements iii  

1 Introduction 1  
1.1 Problem Definition and Motivation 1  
1.2 Overview 3  

2 Web Service Composition: Techniques and Tools 4  
2.1 Introduction to Web Services 4  
2.2 Web Service with Semantics 6  
2.3 Web Service Composition Framework 7  
2.4 Discussion on the Framework 8  
2.5 Web Service Composition Tools using Workflow 9  
2.5.1 Service Composition and Execution Tool (SCET) 10  
2.5.2 Adaptive and Dynamic Composition with eFlow 10  
2.6 Web Service Composition using AI Planning 11  
2.6.1 Rule-Based Composition—SWORD 12  
2.6.2 Hierarchical Task Network based Composition—SHOP2 13  
2.7 Summary 14  

3 Web Service Composition Using SHOP2 Planner 15  
3.1 Australian Tourism Data Warehouse (ATDW) 15  
3.1.1 ATWS Request 16  
3.1.2 ATWS Response 17  
3.1.3 SOAP Messages for ATDW Request and Response 17  
3.2 SHOP2: The Technical Details 19
3.3 Case Study: Visiting Perth ................................. 22
3.4 The Design of the Framework ................................. 22
3.5 Summary ..................................................... 24

4 The Implementation ........................................ 25
4.1 WSDL Description of ATDW ................................. 25
4.2 Inside SHOP2: The HTN Planning ......................... 28
4.3 The Implementation in Details ......................... 31
4.4 Summary ..................................................... 34

5 Example Execution Results ............................... 35
5.1 JSHOP Domain and Problem Definitions .................. 35
5.2 parser1 ....................................................... 36
5.3 The Web Service Client ..................................... 37
5.4 parser2 ....................................................... 39
5.5 Summary ..................................................... 41

6 Conclusion and Future Work ............................... 42
6.1 Conclusion ..................................................... 42
6.2 Future Work and Open Problems ......................... 43
   6.2.1 Industry Implementation of Web Services ........... 43
   6.2.2 AI Planning: The SHOP2 Planner ..................... 43
   6.2.3 World-Altering vs. Information-Providing Web Services  45
   6.2.4 The Implementation of Other Web Services ........... 45

A Original Honours Proposal ............................... 46
   A.1 Background ................................................. 46
   A.1.1 Web Service Composition .............................. 46
   A.1.2 AI Planning ............................................. 47
   A.1.3 Problems and Motivations .............................. 47
   A.2 Research Aim .............................................. 48
   A.3 Methodology .............................................. 48
List of Figures

2.1 The Web service architecture adapted from Gottschalk [14] . . . 5
2.2 Web service composition framework [24]. .......................... 7
2.3 The modified Web service composition framework. .............. 8
2.4 An example of a composite service in eFlow’s process schema adapted from [11]. .................................. 11

3.1 ATDW request and response as input and output. ............... 17
3.2 The design of Web service composition architecture using SHOP2 and ATDW. ........................................ 23

4.1 The components of WSDL [15]. .................................... 26
4.2 An example of action decomposition for Visiting Perth. ........ 29
4.3 An example of total-order plans for Visiting Perth. .............. 31
4.4 The detailed design architecture of Web service composition using JSHOP. ........................................ 32

5.1 The invocation of JSHOP using the command prompt. .......... 36
5.2 The composition plan generated by JSHOP in a text file after the invocation. ........................................ 36
5.3 The regular expression that is used as input to parser1 ........... 37
5.4 Stub communication model for Web Service Client (adapted from [26]). 39
5.5 Running parser2 in command prompt and its output. .......... 40
CHAPTER 1

Introduction

The Web is no longer only an information repository, but evolving towards a virtual environment for business process integration. This vision is realized by many of Web services available for interactive business purposes. A Web Service is a software system designed to support interoperable machine-to-machine interactions over the Web [9]. Interoperable means that Web services are operable and composable regardless of the programming languages, the platform, and the communication protocol used [13]. Online banking, flight booking, temperature control, hotel reservations, online bookshop, etc. are examples of Web services that are available and ready for client consumptions. Web service has created enormous industry commitment because of its potential for improving the way we do business online [39].

1.1 Problem Definition and Motivation

According to Gartner Inc. review [1, 17], a survey on 111 companies in the U.S. shows that 65% of the companies are already working on Web service projects or they are considering implementing the services very soon. According to the survey report, these companies still engaging in Web service projects despite the economic slowdown in 2003. The survey also estimated that $3 billions worth of Web service projects have been carried out in 2003. By 2008, it will increase to $15.8 billions. However, the developed Web services are individual, standalone services termed as atomic services. As the services grow rapidly on the Web, the clients’ needs for achieving more complex tasks increase. Web service composition is seen as a new way of accessing or consuming the services online. Service composition is a powerful key promise of service-oriented programming paradigm. With service composition, not only can we consume a single atomic Web service, we can now integrate existing services together to perform more complex tasks. One of the most promising domain for such integration is in tourism, where we already have access to many Web services. For example, flight booking,
hotel reservation, road map, car rental, etc. These atomic Web services could be composed together to improve the process of planning and arranging for holidays.

Of course, composing Web services is not as easy and simple as it may seem. This is due to the complexities in the Web services domain. The complexities lie in describing the Web services so that they are composable, specifying the flow, i.e. the plan, for composition, and coping with the dynamic changes of Web services through time. So far, there are two approaches for Web service composition—the industry approach and the semantic Web approach [32]. The industry describes Web services in Web Service Description Language (WSDL)\(^1\) and uses flow control languages to specify the flow of Web service composition. Example of such languages are like WSFL, XLANG and BPEL4WS [31, 32] for normal Web services. Whilst semantic Web services are described by languages such as DAML-OIL [37] and OWL-S [16] as discussed more in Chapter 2. The semantic Web community resorted in using AI planning for generating composition plans to achieve the automation of plan generation. Automation here means that there is less, or perhaps no user interference in generating plans for composition. Unfortunately, a large amount of current available Web services are described in WSDL. Semantic Web services are still yet to come. So far, they are still under research and development and have not yet been deployed, in a reasonable scale, in real-world.

Recent works in Web service composition only proposed how AI planning can be used for semantic Web service composition. No work on the actual implementation has been reported. This is largely to do with the lack of available semantic Web services. Works detailing the pros and cons of using AI planning for service composition are also scarce. On the other hand, normal Web services are relatively widely available, especially with the support from the Australian Tourism Data Warehouse (ATDW). We can test out the AI planning ideas on normal Web services. So far, there is no reported work on implementing automated service composition using AI planning with normal, WSDL-based Web services. Therefore, our aim is to use AI planning to compose normal Web services. In this project, we use normal Web services offered by the ATDW. These Web services are called Australian Tourism Web Services (ATWS). We propose a general framework which automatically generates composition plans using AI planning for composing services described in WSDL. Services offered by ATDW (ATWS) are used as examples in the case study. This framework uses an AI planner called Simple Hierarchical Ordered Planner (SHOP2) to generate the composition plan. A prototype is developed to show the feasibility of such framework. Lessons learned from applying AI planning in Web service composition are also

\(^1\)Many acronyms are used in the Web Service domain, so in order to ensure the clarity of this document, a glossary is provided in Appendix B.
1.2 Overview

Some previous works in Web service composition including the tools are reviewed in Chapter 2. In Chapter 3, we introduce the building blocks of the framework. These include the ATWS, as well as SHOP2. We also provide a case study to illustrate the Web service composition process. The framework design and implementation are discussed in depth in Chapter 4. The example execution results are presented in Chapter 5. Finally, to conclude our work, we identify the limitations and open problems in using AI Planning techniques for Web service composition. Suggestions for future works are also presented in Chapter 6.
CHAPTER 2

Web Service Composition: Techniques and Tools

Web service is the current trend of doing business online. It supports e-Business, e-Commerce, and a variety of service-oriented applications that are available on the Web for clients to consume. Simple examples of Web services include Amazon Web services at Amazon.com and the search and spelling checking services provided by Google. Most of the services available are *atomic services*. They provide a single, stand-alone service for clients. However, the need for more complex services is increasing in order to accomplish more complex tasks. Therefore, it is important to reuse the atomic services by combining or composing them together to meet the demand. In this chapter, we will look at what Web service composition is and its supporting technologies.

2.1 Introduction to Web Services

Web services are applications accessible via the World Wide Web to be consumed by clients. Clients of a Web service can be either human users or other applications, such as autonomous agents, that consume the services. A client is usually referred as a *service requester*.

Figure 2.1 illustrates the architecture of Web services, which consists of three major entities—the service requester, service provider, and service registry. In a typical Web service scenario, for example a flight booking service, a service requester will request the price of desired flight from the service provider by sending a message to the provider. Once the provider receives the message, it then processes the message and sends back the response, which in this case the price of the flight requested. This communication, often called binding, is realized by SOAP. Web services can be located by the service requester from a registry, for example the UDDI. UDDI is a repository where service providers register their services using WSDL. SOAP and WSDL are the technologies standardized
by the World Wide Web Consortium (the W3C) for Web service applications. In short, these standards can be defined as [2]:

- Simple Object Access Protocol (SOAP)—a standard communications protocol for Web services.
- Web Service Description Language (WSDL)—a standard mechanism to describe Web services.
- Universal Description, Discovery, and Integration (UDDI)—a standard mechanism to register and discover Web services.

The primary reason for the standardization is to realize the main goal of Web services, i.e. interoperability [28]. This was fundamentally supported by the key technology shared by the standards, which is the Extensible Markup Language (XML) that enables message exchange among any applications [2]. Different organizations may build their Web services using different programming languages and the services may run on different platforms too. Hence, it is crucial to have an architecture that provides a standard means of interoperating between those applications [9]. This important property allows Web services to be developed using any programming language and deployed on any platform thus giving Web services an advantage over other Web-based applications.
2.2 Web Service with Semantics

The previous section introduced Web service and its architecture. A Web service description is given in WSDL, which syntactically describes the service interface. The description of Semantic Web services, however, is described semantically. According to Harmelen and Horrocks [37], the aim of the Semantic Web is to introduce a Web with which both humans and machines can communicate. Humans are good at inferring different senses a syntactical word represents based on its current context. However, a computer program needs extra information for disambiguating one word sense from another. It requires a way of representing information so that its meaning will be machine-processable or machine-accessible.

Basically, ‘semantically’ means representing information in such a way that a machine can understand its meaning, which then allow users to search for services that meets their needs and requirements. To make this vision a reality, Semantic Web needs languages to define ontologies. Ontology is to define the relationship between concepts not words. For example, ‘fly’ could means a flying verb or an insect. ‘Hotel’ and ‘motel’ is different but they both represent the concept of ‘accommodation’. Languages such as Resource Description Framework (RDF), Resource Description Framework Schema (RSDF) [18], Ontologies Inference Layer (OIL) [37], and DARPA Agent Markup Language (DAML) [16] family of languages such as DAML-S (currently called OWL-S) and DAML+OIL are being used. These languages handle ontologies to describe semantics to the Web. They have a well-defined semantics and that enable the markup and manipulation of complex taxonomic and logical relations between entities on the Internet [20]. Semantic allows the description of the domain concepts and the relationships between concepts and services such as the previous fly and hotel-motel examples. While WSDL provides a description of a Web service in terms of its inputs, outputs, location of the service, and its operation, the Semantic Web provides the description of the service in terms of its functional information and models the pre-conditions and post-conditions of a particular service [32].

However, semantic Web services are not yet as widely available as the ‘normal’ Web services discussed previously, Semantic Web services are still under development as much research is still going on in the area. Therefore, we focus on the ‘normal’ Web services in our implementation of Web service composition. A detailed discussion of the implementation is presented in Chapter 3 and Chapter 4.
2.3 Web Service Composition Framework

The composition of Web services becomes essential when there is no single service that could perform a certain task, but certain combination of available Web services could. Chandrasekaran [12] explained Web service composition as the ability to take (re-use) existing services and combine them to form new services. Rather than having one service at a time, we could compose services, if necessary, to achieve our goals and complete more complex tasks. According to Ponnekanti et al. [23], the composition of Web services can reduce the development time and effort for new application as it allows software re-use.

Figure 2.2 illustrates the framework for Web service composition. The general idea of the framework is to model the participants and processes involved in Web service composition. The participants are the service requester who is a user that wants to consume Web services and the service provider who provides the services. The provider will need to register his/her services in the service repository. An example of a repository is the UDDI. The translator translates users specification (external specification) into the internal language understood by the process generator. For every request made by the service requester, the process generator will generate a plan for Web service composition that suites the specification and can fulfill the request. The plan generated composes Web services available in the service repository. If there are more than one plan for a request, the evaluator will evaluate the plans and proposes the best one. The execution engine executes the plan and returns the results to the service requester.
2.4 Discussion on the Framework

Although in our implementation we only focus on the process generator, another component seems to be necessary for improving the reusability. This component links the translator and the process generator. This component can be called flow control language that specifies the flow languages to facilitate the generation and execution of plans by the process generator. Figure 2.3 shows the Web service composition in a much general context, where Web services may be defined using many different languages. The Web services are of different languages and hence the Translator needs to translate the languages to the language used by the Planner. The Planner will need the Flow Control Language to specify its generated composition plans. The Flow Control Language component is linked back to the Translator for translation of the flow language to the Web services’ languages. This is necessary for execution of the composition plans. Note that the Planner could be built in different languages too. Hence, the Translator should be able to translate \( n \times m \) languages for a general framework to cope with the number of Web services languages (n), and the number of Planner languages (m).

The flow control languages include BPEL4WS, XLANG, WSFL, BPML, and WSCI. These languages are being used to support business interactions that require long-running interactions driven by an explicit business model [35]. Another reason for using these languages is to enable states for processes to be recorded. These states are more complex than a simple request response in order
Web Service Composition Languages aim at combining Web services together in a process-oriented way. These languages use one or more WSDL services which are then combined together to provide composite services. Hence, these languages provide the means to specify a process model that describe the order of execution of the composite services. The same way applies to Semantic Web services where services are described semantically in DAML-S and more recently, OWL-S.

However, the composition tasks are complex and human are unable to deal with this complexity manually for the following reasons:

- The number of services available on the Internet is increasing dramatically as Web service gain enormous attention from the industry. The search for Web services suitable for composition is, of course, hard and time consuming to deal with manually.

- Existing Web services can be updated and new Web services can be created at run time. A service repository should be able to dynamically track and cope with the changes as we want our decision on composing the services based on the up-to-date information.

- Web services are developed by different organizations that use different languages to describe the services offered. There are no standard language for this purpose and that creates the complexity for composing them together.

With the complexity that arises in Web service composition domain, it seems impossible for humans to search for services, generate plans, and execute them manually. Therefore, we need an automatic or semi-automatic composition tool to assist us in composing the services.

2.5 Web Service Composition Tools using Workflow

Web service composition tools are tools that can help us to combine possible Web services together for certain tasks. The tasks are of complex type and need multiple services to achieve their goals. Hence, the tools provide a way of automating (or semi-automating) Web service composition and they fall into two categories—static composition and dynamic composition. The static composition chose Web services to be composed at design time while the dynamic composition chose the services at run-time. Examples of static composition tools are eFlow and SCET which use workflow method to model their composition plans.
According to Srivastava and Koehler [33], Web service composition workflows are derived from business model. The composition in workflow can be divided into two types—static and dynamic. Static Web service composition requires the service requester to build a process model that consists of tasks and data dependency before the planning composition starts. Hence, the services are chosen at design time. Each task contains a query used to search atomic Web services. A process model is commonly specified in a graph. Dynamic service composition, however, generates the process model and selects the services automatically where they are chosen at run-time. The service requester just specify some constraints including the dependency of atomic services and his/her preferences [24].

2.5.1 Service Composition and Execution Tool (SCET)

SCET is a static composition tool that composes Web services and store them as WSFL based specifications. SCET consists of four components—the process designer, simulation model generator, Perl execution code generator, and an execution monitor using Java RMI server. The process designer is used to design the layout of the process structure and provides information about activities and links used in the process. An activity node stores information about the Web services that implement it including the services’ WSDL specifications. The links are divided into two types—the control links and the data links. The control links model the control flow whilst the data links model how the output of one activity is linked to the input of another activity. The Perl execution code is automatically generated from the WSFL based specifications for easier execution of composed Web services [12].

Chandrasekaran [12] also claims that SCET can be modified to be a dynamic composition tool. From our viewpoint, for SCET to be a dynamic tool, instead of supporting just WSDL, it could be modified to implement BPEL4WS that supports a blend of graph and construct based process modeling. This is because BPEL4WS provides various supports for Web service composition such as support for executable and abstract business processes, and support for robust transactions and exceptions handling mechanism [22].

2.5.2 Adaptive and Dynamic Composition with eFlow

Another example tool using workflow technique is Hewlett Packard’s eFlow. eFlow is a dynamic and adaptive tool where a composite service is described as a process schema that composes other services. This composite service is modeled in a graph of flow structure which specifies the order of execution of
the nodes in the process. A service node represents the invocation of a basic or composite service. A service node specification includes the definition of data that the node is allowed to read or modify, and the description of the service to be invoked.

In figure 2.4, the rounded boxes represent invocations of single or composite services. The black circle represent the starting and ending point of a process. The horizontal lines represent parallel invocation of Web services and synchronization after service executions. This lines are one of eFlow’s decision node type. The definition of a service node contains a search information that is used to query the service [24]. This gives the dynamic feature of eFlow which lies in its ability to discover Web services dynamically. Other dynamic features offered by eFlow are dynamic service node creation and dynamic service process modification. eFlow also allows two ad-hoc changes—process schema modifications and process instance state modifications which are explained in detail in [11]. Those features give eFlow its dynamic and adaptive properties.

2.6 Web Service Composition using AI Planning

AI planning is more towards dynamic composition tools, which composes the services at runtime, such as SHOP2 and SWORD. Both tools incorporate AI Planning techniques to generate their composition plans. In AI planning, the plans are generated automatically [24].
AI Planning techniques that could be implemented in service composition include Situation Calculus, Graphplan, Constraint Satisfaction Problem (CSP), and Constraint Programming [21]. The planning problem is viewed as generating a sequence of operators that will transform the current state of the environment into a goal state. Russel and Norvig [25] defined the problem of planning as one type of problem solving where an agent uses its beliefs about available actions and potential outcomes before it can identify a solution from an abstract view of possible plans. What makes AI Planning possible for Web service composition is the description of Web services in the DAML-S or OWL-S which describes the services in a more machine friendly manner [21].

Even though AI Planning is seen to be more suitable for Semantic Web service composition, it could also produce reasonable results for normal Web service with the assumption that the component services share common terminologies. As for our implementation, we will compose Web services described in WSDL and use AI Planning technique for the generation of composition plans.

2.6.1 Rule-Based Composition—SWORD

SWORD is a composition tool that allows basic services to be composed quickly by using keyword-based searching. The plan generation for service composition is realized by the rule-based technique. It does not require deployment of Web service standards such as WSDL, SOAP, RDF, or DAML though it could benefit from them. In SWORD, services are defined in a “world model” based on their pre-condition and postcondition which are inputs and outputs respectively. The world model is an Entity Relationship(ER) model, which consists of entities and relationships among them. However, instead of using ER model like the traditional data modeling, SWORD uses it to describe the inputs and outputs of the Web services. Given the inputs and outputs of the services, a rule-based system is then defined for indicating which inputs produce which output through a sequence of services [23].

Generally, SWORD cannot handle services with multiple side effects, i.e. credit/debit of a bank account. Instead, it is only suitable for information providing Web services [23] that does not alter the state of the ‘world’. However, in our opinion, a good plan generator should be general enough to cope with either information providing or world altering Web services regardless of what languages are being used by the Web services. Moreover, SWORD can generate uncertain results if the pre-condition (the input) fail to uniquely determines a post-condition (the output) [24].
2.6.2 Hierarchical Task Network based Composition—SHOP2

Differs from the above tools, SHOP2 uses Hierarchical Task Network (HTN), an AI Planning technique with OWL-S giving semantic to its application. HTN planning generates plans by task decomposition (divide problems into subtasks). Its objective is to create a sequence of actions that perform tasks for users. In SHOP2, the plans are generated based on the order of execution of the tasks given the knowledge about the domain. This means SHOP2 knows the current state of the world at each step in the planning process. Having the knowledge of the current state, SHOP2’s pre-condition-evaluation mechanism is able to include inferencing and reasoning power plus its ability to call external programs. SHOP2’s knowledge base consists of operators and methods, which includes non-action but related facts and axioms. An operator is an action that describes what to be done to achieve a primitive task. A method contains ways to decompose some tasks into partially ordered subtasks. Planning problem for SHOP2 is given by (S, T, D) where S is the initial state, T is a task list, and D is the domain description. SHOP2 takes this as its input and returns a plan, P = (p_1 p_2 p_3...p_n). P consists of a sequence of operators that will achieve T from S in D.

The planning process involves encoding the OWL-S Web services descriptions into SHOP2 planning domain (D). For that purpose, we have to translate the OWL-S description into the domain (D). This requires a number of processes to be carried out as explained in details in [27]. Hence, SHOP2 is a composition tool for Semantic Web services. In the context of Semantic Web services, OWL-S, originated from DAML-S [19], is used to describe the services and provide language constructs to specify the flow of the services. In other words, OWL-S provides descriptions in two folds, one is service description (corresponds to WSDL used by normal Web services); the other is service control flow description (corresponds to BPEL4WS and WSFL in normal Web services). The SHOP2 planning process and the OWL-S description parsing and generation process are not tightly coupled. The SHOP2 as a planner takes input operators generated by parsing OWL-S service descriptions and produce composition plans that are in turn translated into OWL-S description. Because of the relevant similar functionality between OWL-S and WSDL, based on the assumptions that unambiguous descriptions are used WSDL, we can use SHOP2 to generate plans based on the operations described by WSDL. Basically, SHOP2 is used as a plan generator independent of the service description languages. These plans can be used to specify the order of Web services execution.
2.7 Summary

As Web service proves essential in e-business, much work has been done to improve the technology. It includes developing and improving its standards, adding semantic to Web services through annotation, and developing tools to compose Web services. Different approaches have been implemented in developing the tools including rule-based approach, planning-based approach, and using flow structure to derive the composition and execution plan of Web services. AI planning has the potential to be used as an automated process for generating plans for Web service composition as it is viewed to be more intuitive and provide more flexibility in composing Web services.
CHAPTER 3

Web Service Composition Using SHOP2 Planner

As Web services emerge in many areas, tourism has shown an exponential growth of Web services offering services like hotel reservation, flight booking, car rental online, and many more. To have a real-world example of Web service composition using AI Planning technique, we will consider a simple example in tourism domain for the case study. In this chapter, we will discuss in details about the Web services offered by the Australian Tourism Data Warehouse (ATDW), the technical side of SHOP2, and the design of how to compose Web services using SHOP2. This will lay the foundation of future chapters.

3.1 Australian Tourism Data Warehouse (ATDW)

The Australian Tourism Data Warehouse (ATDW) is developed to encourage and assist the development of tourism in Australia. This is pioneered by the Australian Tourism Organization, the Australian States and Territories, and Tourism Australia [3]. The aim is to position Australia as a major competitor in global e-Economy by increasing the number and expenditure of visitors to Australia. ATDW is a system for storing information on Australian tourism products. It is a central repository where the information is compiled in agreed format and electronically accessible by authorized people who want to include it in their Web sites.

ATDW offers tourism Web services (ATWS) for providing information, accommodation, transport, events, destinations, attractions, hire, and tours. It is designed to cater for the need to market a comprehensive range of Australian tourism product and destination information regardless of geographical limitations as it is an online application. It is hoped that the ATDW can help in broaden the distribution of the economical benefits of tourism in Australia.
3.1.1 ATWS Request

As mentioned previously, the ATDW stores other Web services known as the Australian Tourism Web Services (ATWS). To consume a service, we have to invoke ATWS. For that, we have to provide a SOAP request message as its input. The SOAP used in this application is of version 1.1. The interface to the ATWS is defined in WSDL. The functions exposed by WSDL are Query Set 1 and Query Set 2. However, Query Set 2 has become the dominant approach to query the ATWS. In Query Set 2, the functions are not exposed individually. They can be accessed by using a wrapper function called the CommandHandler. The query functions of Query Set 2 are:

- QueryProducts
- QueryProductsNextPage
- GetProduct
- Get ProductService
- GetCities
- GetSuburbs
- GetCountryStateArea
- GetDomesticRegionArea
- GetAttributesInLocation
- GetAttributeDefinitions

The CommandHandler accepts three input parameters as defined in WSDL. Each of the input parameters are treated as a String. They are the Distributor Key, the Command Name, and the Command Parameters. The Distributor Key is the key issued to authorized people who registered with the ATDW to gain access to the ATWS. The Command Name is the name of the Query Set 2 function that we want to invoke. The Command Parameters are the input parameters specifically to the Command Name. This input parameters, though written in XML, are treated as a single string. However, it is crucial to write it in a correct, valid format of XML. Figure 3.1 illustrates the process of invoking ATWS.
3.1.2 ATWS Response

Once the ATDW is invoked, the response will be returned to client in the form of XML. However, the output is not returned as separate XML element but it is returned as a string value. The results can be validated with the XML Schema file provided in the ATDW Distributor Kit.

3.1.3 SOAP Messages for ATDW Request and Response

To invoke ATDW for ATWS, we need to use a SOAP message. SOAP is an abbreviation of Simple Object Access Protocol which is a communication protocol that specifies the format for exchanging messages over the Web. It is purely XML-based protocol which is designed for a distributed, decentralized environment. This property gives SOAP the interoperability to be used in a heterogeneous environment [29] especially when dealing with Web services. That means exchanged messages can be regardless of the communication protocol that we are engaging with.

To request, we send a SOAP message from our application to the ATWS. Once it is received and processed, the ATWS will send another SOAP message as a response. Both request and response messages are written in XML in a standard format which is understood by both parties. The following is a simple example of SOAP request and response messages respectively:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
```
And the response message is as follows:

```xml
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.flight-bookings.com/prices">
    <m:GetFlightPriceResponse>
      <m:Price>745.00</m:Price>
    </m:GetFlightPriceResponse>
  </soap:Body>
</soap:Envelope>
```

The above example shows a sample request and response messages expressed in SOAP. It is simply requesting a price for a flight to Kuala Lumpur on 15th July 2005. The response message then returns the price of a given flight name, which is the Malaysian Airlines (MAS). Note that the actual message is within the SOAP body element. They are called the application-specific elements and they are not part of the SOAP standard. Different application may have different XML namespace (xmlns) and XML tags.

Note that the above example is just to give us the flavour of how the request and response messages will look like in SOAP. Appendix C shows a sample of SOAP request and response messages to ATDW. Even though SOAP is a standard way of communicating with ATDW, we could also use other means of communicating the messages over to ATDW via using either HTTP POST, HTTP GET, or MIME. ATDW service interface are being described in WSDL and is discussed in Chapter 4.
3.2 SHOP2: The Technical Details

SHOP2 is written in Lisp programming language. To use SHOP2, the available services should be translated into operators and described in domain definition file. The composition problem should be translated and described in a separate problem definition file. Both files are then used with the SHOP2 engine to generate plans. To do so, we have to understand the technical components of SHOP2 terms, axioms, operators, and methods. In SHOP2, a term can be one of the following:

- a variable
- a constant
- a number
- a list-term
- a call-term
- an eval-term

A list-term is of the form

\[(\text{list} \ t_1 \ t_2 \ldots t_n \ [. \ 1])\]

where list is an optional reserved word and \(t_i\) is a term which specifies the item of the list. [. 1] is also optional and should it appear in the list, item 1 should evaluate to a list; all items in 1 are included in the list after \(t_1\) to \(t_n\).

A call-term is of the following form:

\[(\text{call } f \ t_1 \ t_2 \ldots t_n)\]

where \(f\) is a function and \(t_i\) is a term. Call-term indicates SHOP2 that the \(f\) is a procedure. Hence, SHOP2 needs to apply the function \(f\) on each of \(t_i\). Different from the call-term, an eval-term took the form of:

\[(\text{eval } \text{general-lisp-expression})\]

where the expression is evaluated. Any variable that exists in the expression will be replaced by the value that it is bound to. For example, let \(?x = 10\). Hence, the term:

\[(\text{eval } (\text{list} \ (* \ ?x \ 2) \ (+ \ ?x \ 12)))\]

will yield \(20 22\) as the answer.
SHOP2 requires a planning domain and a single problem or a problem set as its inputs. The planning domain consists of operators, methods, and axioms. In SHOP2, an operator took the following form:

(:operator h pre del add [c])

where

- **h** is the operator’s head, a primitive task atom which normally begins with an exclamation mark.
- **pre** is the pre-condition in logical expression form.
- **del** is a delete list for which its element could be any of the following:
  - a logical atom
  - a protection condition
  - an expression
- **add** is an add list of logical atoms that has the same form as in del.
- **[c]** is the operator’s cost. If it is omitted, the cost is 1.

A SHOP2 method is in the following similar form:

(:method h [n1] C1 T1 [n2] C2 T2 ...[nk] Ck Tk)

where

- **h** or head is a task atom in which no call- or eval-terms can appear.
- **C_i** is the pre-condition in a logical expression form.
- **T_i** is called the method’s tail which is a task list. It can contain call-terms.
- **n_i** is the name for C_i T_i pair. These names are optional and it can be omitted. If so, a unique name will be assigned for the each pairs.

Tasks specified in the method’s head can be executed by executing the tasks in one of its tails given that that particular task’s pre-condition is met. The following example considers cutting a piece of paper either with scissors or a blade:

(:method (cut-paper ?paper)
  (have-scissors ?scissors)
  ((!cut-with-scissors ?paper ?scissors))
(have-blade ?blade)
((!cut-with-blade ?paper ?blade))

This method shows how the pre-condition affect the end result. If the pre-condition (have-scissors ?scissors) is satisfied, then the task ((!cut-with-scissors ?paper ?scissors)) will be executed and likewise.

An axiom in SHOP2 is an expression of the form:

\[ \text{(:- a [name}_1 \text{ E}_1 \text{ [name}_2 \text{ E}_2 \ldots \text{[name}_n \text{ E}_n}) \]

where a is the axiom’s head and all of the [name] \text{ E}_i are its tail. [name] is the name of the expression \text{ E}_i and it is optional. Although these names does not have any semantic meaning to SHOP2 but they could aid the debugging process by the user. The meaning of the axiom is that a is true if one of the \text{ E}_i is true when others are false.

SHOP2 needs to know the domain of the problem before it could generate some plans as output. To define the domain in SHOP2, we use the following syntax:

(defdomain domain-name (i_1 i_2 ... i_n))

where domain-name is a symbol which tells the name of that problem domain. i_i could be either an operator, a method, or an axiom.

The planning problem in SHOP2 is of the form:

(defproblem problem-name domain-name (a_1 a_2 ... a_n) T)

where both problem-name and domain-name are symbols, a_i is a logical atom, and T is a task list. The problem is defined such that it could be solved by addressing the tasks in T with the initial states which are defined by each of a_i.

The planning problem could also be defined as multiple planning problem in a planning problem set, which is written in the form:

(def-problem-set set-name (p_1 p_2 ... p_n))

where set-name is a symbol representing the problem set name and p_i is the name of a planning problem.

To execute SHOP2 planning process, we use either find-plans, which finds plans for a single planning problem, or do-problems, which finds plans for a set of planning problem. SHOP2 produces its output as a plan. A plan is a list of the form:

(h_1 c_1 h_2 c_2 ... h_n c_n)