UNIVERSITY OF CRETE

COMPUTER SCIENCE DEPARTMENT

HWSC - A TOOL FOR WEB SERVICES COMPOSITION

TZAGARAKIS CHARALAMPOS

Master's Thesis

Heraklion, November 2012

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

HWSC - Εργαλείο Σύνθεσης Ηλεκτρονικών Υπηρεσιών

Εργασία που υποβλήθηκε από τον

Χαράλαμπο Τζαγκαράκη

ως μερική εκπλήρωση των απαιτήσεων για την απόκτηση

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΕΙΔΙΚΕΥΣΗΣ

Επόπτης Καθηγητής: Δημήτρης Πλεξουσάκης
Καθηγητής
Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

Η εργασία αυτή πραγματοποιήθηκε στο Πανεπιστήμιο Κρήτης στο Τμήμα Επιστήμης Υπολογιστών

ABSTRACT

Web services can be used in a variety of applications, ranging from household tools to video games, assisting users in solving different types of problems. However, the most significant impact is witnessed in businesses that use Web services for commercial and integrated business solutions.

A Web service is typically characterized by two sets of parameters: a set of inputs (usually represented by a SOAP request) and a set of outputs (usually represented by a SOAP response). A successful execution of a Web service with all required input parameters (which are assumed to be available) is expected to produce the required output parameters. Given a request containing a set of input parameters and requesting a set of output parameters, a typical problem is to find candidate single Web services that can produce the required output if the required input is provided. This problem is defined as Web Service discovery in literature.

In this thesis we focus on the case where there is not any single Web service that fully satisfies a given request. In this case we must compose multiple Web services in order to satisfy the given request. Web Service composition aims to address that problem. Web Service composition involves combining and coordinating a set of services in a workflow with the purpose of achieving functionality that cannot be realized with any individual service. Several service composition models have been proposed, with the most prominent ones being service orchestration and service choreography.

We present a tool named HWSC which reads a request expressed by a use in a predefined format and attempts to answer the request by creating a composite service combining Web services from a repository. The tool first reads the WSDL descriptions of the Web Services contained in the repository and then applies a composition algorithm that results in a composite service that satisfies the request. The innovative characteristics of the tool are: 1) Provides a lightweight approach to composition based only on WSDL input-output descriptions. 2) Uses similarity techniques while reading a user request allowing for a 20% probability of error in input and output names. 3) Uses an algorithm based on STRIPS, an automated planner in artificial intelligence, in order to find the composition sequence that satisfies the user request. 4) Exports the final composition sequence in a BPEL template file, which can be edited by tools supporting BPEL and executed by a BPEL engine.

Supervisor: Dimitris Plexousakis
Professor
Computer Science Department
University of Crete

ΠΕΡΙΛΗΨΗ

Οι ηλεκτρονικές υπηρεσίες μπορούν να χρησιμοποιηθούν σεμια ποικιλία εφαρμογών, που κυμαίνεται από εργαλεία οικιακής χρήσης μέχρι βιντεοπαιχνίδια, βοηθώντας τους χρήστες για την επίλυση διαφόρων τύπων προβλήματα. Ωστόσο, η πιο σημαντική χρήση τους είναι σε επιχειρήσεις οι οποίες χρησιμοποιούν τις ηλεκτρονικές υπηρεσίες για εμπορικούς σκοπούς και για ολοκληρωμένες επιχειρηματικές λύσεις.

Μια ηλεκτρονική υπηρεσία τυπικά χαρακτηρίζεται από δύο σύνολα παραμέτρων: ένα σύνολο των εισόδων (συνήθως αντιπροσωπεύεται από μια αίτηση SOAP) και ένα σύνολο εξόδων (συνήθως αντιπροσωπεύεται από μια απάντηση SOAP). Η επιτυχής εκτέλεση μιας ηλεκτρονικής υπηρεσίας με όλες τις απαιτούμενες παραμέτρους εισόδου (οι οποίες υποτίθεται ότι είναι διαθέσιμες) αναμένεται να παράγει τις απαιτούμενες παραμέτρους εξόδου. Λαμβάνοντας υπόψη ότι ένα αίτημα περιέχει ένα σύνολο παραμέτρων εισόδου και ζητάει ένα σύνολο παραμέτρων εξόδων, ένα τυπικό πρόβλημα είναι να βρεθεί μία υποψήφια ηλεκτρονική που μπορεί να παράγει την απαιτούμενη έξοδο εάν η απαιτούμενη είσοδος παρέχεται. Αυτό το πρόβλημα ορίζεται ως ανακάλυψη Υπηρεσία Web στη βιβλιογραφία.

Σε αυτήν την εργασία επικεντρωνόμαστε στην περίπτωση κατά την οποία δεν υπάρχει καμία ενιαία ηλεκτρονική υπηρεσία που ικανοποιεί πλήρως ένα συγκεκριμένο αίτημα. Σε αυτή την περίπτωση θα πρέπει να γίνει σύνθεση πολλαπλών ηλεκτρονικών υπηρεσιών , προκειμένου να ικανοποιηθεί το συγκεκριμένο αίτημα. Η σύνθεση ηλεκτρονικών υπηρεσιών έχει ως στόχο να αντιμετωπίσει αυτό το πρόβλημα. Η σύνθεση ηλεκτρονικών υπηρεσιών περιλαμβάνει το συνδυασμό και τον συντονισμό μιας σειράς υπηρεσιών σε μια ροή εργασίας με σκοπό την επίτευξη λειτουργικότητας που δεν μπορεί να πραγματοποιηθεί με οποιαδήποτε μεμονωμένη υπηρεσία. Αρκετά μοντέλα σύνθεσης ηλεκτρονικών υπηρεσιών έχουν προταθεί, με τα πιο γνωστά να είναι η ενορχήστρωση υπηρεσιών και η χορογραφία υπηρεσίας.

Θα παρουσιάσουμε ένα εργαλείο το οποίο ονομάζεται HWSC το οποίο διαβάζει ένα αίτημα που εκφράζεται σε μια προκαθορισμένη μορφή και επιχειρεί να απαντήσει στο αίτημα, με τη δημιουργία μιας σύνθετης υπηρεσίας που συνδυάζει τις ηλεκτρονικές υπηρεσίες από μια αποθήκη ηλεκτρονικών υπηρεσιών. Αρχικά το εργαλείο διαβάζει τις περιγραφές WSDL των Υπηρεσιών του Παγκοσμίου Ιστού που περιέχονται στο αποθήκη ηλεκτρονικών υπηρεσιών και στη συνέχεια εφαρμόζει ένα αλγόριθμο σύνθεσης που οδηγεί σε μια σύνθετη υπηρεσία που ικανοποιεί το αίτημα. Τα καινοτόμα χαρακτηριστικά του εργαλείου είναι τα εξής: 1) Παρέχει μια βασική προσέγγιση για την σύνθεση που βασίζεται μόνο σε WSDL εισόδου-εξόδου περιγραφές. 2) Χρησιμοποιεί τεχνικές ομοιότητας διαβάζοντας ένα αίτημα του

χρήστη επιτρέποντας του 20% πιθανότητα σφάλματος στα ονόματα εισόδου και εξόδου 3) Χρησιμοποιεί ένα αλγόριθμο βασιζόμενο στο μοντέλο STRIPS , το οποίο αποτελεί ένα αυτοματοποιημένο προγραμματιστή διαδικασιών στην περιοχή της τεχνητής νοημοσύνης, προκειμένου να βρει την ακολουθία σύνθεσης που ικανοποιεί το αίτημα του χρήστη. 4) Εξάγει την τελική ακολουθία σύνθεσης σε ένα template αρχείο BPEL, το οποίο μπορεί να επεξεργαστεί με εργαλεία υποστήριξης BPEL και εκτελεστεί από μια μηχανή BPEL.

Επόπτης Καθηγητής: Δημήτρης Πλεξουσάκης
Καθηγητής
Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω θερμά τον επόπτη μου , Καθηγητή κ. Δημήτρη Πλεξουσάκη για την εμπιστοσύνη που μου έδειξε κατά τη διάρκεια των μεταπτυχιακών σπουδών καθώς για τη πολύτιμη βοήθεια και καθοδήγηση του σε όλο αυτό το χρονικό διάστημα.

Επίσης ευχαριστώ τον Καθηγητή κ. Χρήστο Νικολάου , μέλος της εξεταστικής επιτροπής της μεταπτυχιακής μου εργασίας τόσο για το χρόνο που αφιέρωσε όσο και για τις πολύτιμες συμβουλές του.

Ακόμη ευχαριστώ πολύ τον Καθηγητή κ. Κωνσταντίνο Μαγκούτη για την προθυμία του να συμμετάσχει στην εξεταστική επιτροπή της μεταπτυχιακής μου εργασίας καθώς για τις πολύτιμες παρατηρήσεις του.

Πολλές ευχαριστίες θα ήθελα να εκφράσω σε όλους τους φίλους μου τόσο μέσα από το Πανεπιστήμιο όσο και έξω από αυτό , για τη στήριξη τους κατά τη διάρκεια των μεταπτυχιακών σπουδών.

Τέλος το μεγαλύτερο ευχαριστώ ανήκει δικαιωματικά στους γονείς μου Μανώλη και Μαρία καθώς και στην αδερφή μου Ευφρανσία , για την πολύτιμη και αδιάκοπη στήριξη και βοήθεια τους για να ολοκληρώσω τις μεταπτυχιακές μου σπουδές. Σας ευχαριστώ πολύ !!!

CONTENTS

CHAPTER 1 13
1. WEB SERVICES 13
1.1 WEB SERVICES DEFINITION 13
1.2: A SIMPLE EXAMPLE OF A WEB SERVICE 14
1.3: WHY USE WEB SERVICES 16
1.4: WEB SERVICES TECHNOLOGY 16
1.4.1: SERVICE TRANSPORT 17
1.4.2: SERVICE MESSAGING 17
1.4.3: SERVICE DESCRIPTION 17
1.4.5: SERVICE REGISTRY 18
1.4.6: SERVICE COMPOSITION 19
1.5: APPLICATIONS SCENARIOS 19
1.5.1: BUSINESS-TO-BUSINESS 19
1.5.2: ENTERPRISE APPLICATION INTEGRATION 20
1.6: IMPLEMENTATION SCENARIOS 22
1.6.1: SIMPLE SERVICE 22
1.6.2: COMPOSITE SERVICE 23
1.6.3: INTERIM SERVICE 23
1.6.4: BUS SERVICE 24
CHAPTER 2 26
2. SOA - BASIC MODEL OPERATION OF WEB SERVICES 26
CHAPTER 3 28
3. WSDL - WEB SERVICES DESCRIPTION LANGUAGE 28
3.1 WSDL SPECIFICATION 28
3.2 BASIC EXAMPLE IN WSDL (HelloService.wsdl) 30
3.2.1 EXPLANATION THE DATA OF THE EXAMPLE 31
CHAPTER 4 35
4. BPEL – BUSINESS PROCESS EXECUTION LANGUAGE 35
4.1 WHAT IS A WORKFLOW? 35
4.2 IMPLEMENTATION OF WORKFLOW 36
4.2.1 ACTIVITIES 36
4.2.2: PARTNERS: INTERACTING WITH OTHERS 37

4.2.3: SERVICE LINK TYPE 37
4.2.4: HOW TO INDICATE THE PARTNER IN WORKFLOW 37
4.2.5: FACING THE PROBLEMS AND MISTAKES IN THE EXECUTION 38
4.2.6: LIFETIME OF WORKFLOWS 38
4.2.7: WORKFLOW STARTING 39
4.3: A SIMPLE EXAMPLE OF A WORKFLOW – GET A LOAN 39
4.3.1: STRUCTURE OF THE WORKFLOW 40
4.3.2: DESCRIPTION OF WEB SERVICES THAT WILL COOPERATE FOR YOUR WORKFLOW 40
4.3.3: CREATING THE WORKFLOW 44
4.3.4: INTERACTING WITH THE WORKFLOW 45
4.3.5: OVERALL VIEW OF THE WORKFLOW FOR LOAN APPROVAL 47
4.4: EXTENDING THE PREVIOUS EXAMPLE - ADDING LINKS AND DEALING OF DATA 48
4.4.1: INTRODUCTION 48
4.4.2: CONSTRUCTING THE WORKFLOW 48
4.5: EXTENDING THE PREVIOUS EXAMPLE - ADDING CORRELATION AND FAULT HANDLING 53
4.5.1: INTRODUCTION 53
4.5.2: TAKING THE LOAN: ADDING CORRELATION 53
4.5.3: CORRELATION MESSAGES 57
CHAPTER 5 58
5. WEB SERVICES COMPOSITION 58
5.1 INTRODUCTION 58
5.2 NON-AUTOMATED METHODS AND COMPOSITION LANGUAGES 58
5.2.1: BPEL4WS 58
5.2.2: WSCI 60
5.2.3: WS-CDL 60
5.2.4: WSCL 61
5.2.5: BPML 61
5.2.6: BPMN / XPDL 61
5.3: SEMI-automated SYNTHESIS METHODS 62

5.4: AUTOMATED SYNTHESIS METHODS....63
5.4.1: DESIGN IN HIERARCHICAL TASK NETWORKS....68
5.4.2: DESIGN ACTIONS ESTIMATE REDUCTION....69
5.4.3: DESIGNED USING MODEL CHECKING....70
5.4.4: CALCULUS STATEMENTS....71
5.4.5: DESIGNED USING RULES....72
5.4.6: HYBRID DESIGN....73
5.4.7: PROOF OF THEOREMS....74
5.4.8: TRANSFORMATION AND SYNTHESIS PROBLEM THROUGH CLASSIC DESIGN ACTIONS....75
CHAPTER 6....76
6. WEB SERVICES COMPOSITION WITH HWSC TOOL....76
6.1 INTRODUCTION....76
6.2 WEB SERVICES COMPOSITION PROBLEM FORMALIZATION....76
6.3: HWSC ARCHITECTURE....79
6.3.1: WEB SERVICES DATABASE....79
6.3.2: GOALS XML FILE....82
6.3.3: HWSC ALGORITHM....82
6.3.4: EXAMPLE....88
CHAPTER 7....91
7. EXPERIMENTS....91
7.1: LOAD WEB SERVICES IN MEMORY....91
7.2: O(s) TIME CREATION....92
7.3: TESTING DIFFERENT GOALS....93
CHAPTER 8....95
8. CONCLUSIONS AND FUTURE WORK....95
8.1: CONCLUSIONS....95
8.2: FUTURE WORK....96
CHAPTER 9....97
9. BIBLIOGRAPHY....97

List of Tables

Table 1: Load and save inputs and outputs parameters of Web Services 81
Table 2: Levenshtein table 85
Table 3: Comparison of loading 10 sets of Web Services in memory 92
Table 4: O(s) creation time 92
Table 5: Testing results for 10 goals 93
Table 6: Comparison goals composition time 93

List of Figures

Figure 1:A simple Web Service....13
Figure 2: A Simple Web Service....15
Figure 3: Web Service Stack....17
Figure 4: hub-and-spoke topology of enterprise application integration....20
Figure 5: Web Service topology integration of business applications....22
Figure 6: Simple web service scenario....22
Figure 7: Composite web service scenario....23
Figure 8: Interim web service scenario....23
Figure 9: Bus web service scenario....24
Figure 10: WSDL document sructure....28
Figure 11: HelloService.wsdl....30
Figure 12: The four standard modes supported by WSDL....33
Figure 13: Picture of a Web service that is implemented as a BPEL flow....36
Figure 14: Exterior view of the workflow for a loan getting....39
Figure 15: What's inside the Cloud? Internal view of the workflow for a loan.40
Figure 16: Loan Definitions WSDL (loandefinitions.wsdl)....41
Figure 17: Loan Approver WSDL (loanapprover.wsdl)....42
Figure 18: Loan Approval WSDL (loan-approval.wsdl)....43
Figure 19: Workflow namespaces....44
Figure 20: Definition of partners....45
Figure 21: Definition of variables....45
Figure 22: Receive action in workflow....46
Figure 23: Invoke action in workflow....46
Figure 24: Reply action in workflow....47
Figure 25: Execution the workflow for loan approval....47
Figure 26: Interior view of the loan approval process....48
Figure 27: WSDL description of loan accessor....49
Figure 28: partnerLinkType....50
Figure 29: assessor partner....50
Figure 30: Another receptor....50
Figure 31: Flows and Links....51
Figure 32: Assign action....52
Figure 33: Invoke and reply action....52
Figure 34: Possible cases of the loan approval process....53
Figure 35: Adding actions to present the correlation....54
Figure 36: Using property alias....55
Figure 37: CorrelationSets....57
Figure 38: The parts of a web service description in OWL-S....65
Figure 39: Correspondences between OWL-S and WSDL....66
Figure 40; An example of service in OWL-S....67
Figure 41: Example WSDL files....78

Figure 42: Example STRIPS model 78
Figure 43: States Space of example 79
Figure 44: HWSC main window 80
Figure 45: Path for the Web Services Folder 80
Figure 46: WSDL files in the folder of Web Services 81
Figure 47: Goals file example 82
Figure 48: Compute Distance in similarity algorithm 83
Figure 49: Similarity Algorithm 84
Figure 50: O(s) list creation algorithm 86
Figure 51: Algorithm that exports the final composition sequence of Web Services 87
Figure 52: Repeated procedure for final solution 88
Figure 53: WSDL files of Web Services Example 89
Figure 54: Time loading web services in memory 91
Figure 55: Goals composition time 94

CHAPTER 1

1. WEB SERVICES

1.1 WEB SERVICES DEFINITION

According to the World Wide Web Consortium[1], a web service is defined as a software application identified by a URI, whose interface and bindings can be identified, described and discovered by means of XML language. In addition XML supports direct interactions with other software applications using XML-based messages via protocols based on the Internet.

In simpler words, a web service is a service available on the Internet, which uses a specific messaging system, based on XML and is not tied to any operating system or programming language.

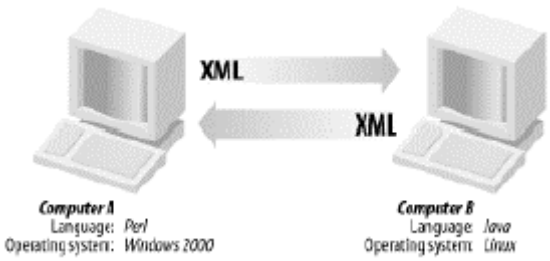


Figure 1: A simple Web Service

It is desirable, although not mandatory, a web service to satisfy 2 additional properties:

- A web service should be self-descriptive. Whenever a web service is published, an interface also should be published (e.g. a description of the web service), so as to enable them to develop integrated systems to integrate with others and be easy to implement. If it is then the SOAP service description can be done using XML grammar and define all public methods, their arguments and return values.

[1] http://www.w3.org/

- A web service should be discoverable. Whenever a web service is created it should be easily discovered. It is actually the process of finding a suitable web service for a given task. The provider of a web service usually augments the service with an interface description that will allow the users to find it and use it.

In summary an integrated web service is a service that:

1) It is available online or in private (intranet) networks.
2) It uses a predefined message system based on XML.
3) It is not tied to any operating system or programming language.
4) It is self-described through a common XML grammar.
5) It is discoverable (using simple and well-known processes to find it).

1.2: A SIMPLE EXAMPLE OF A WEB SERVICE

Figure 2 shows a service provider offering travel agency services. In particular the service provider uses several other web services (provided by other travel agencies, located in various locations (and networks). The provided services are supported for a variety of customers and applications.

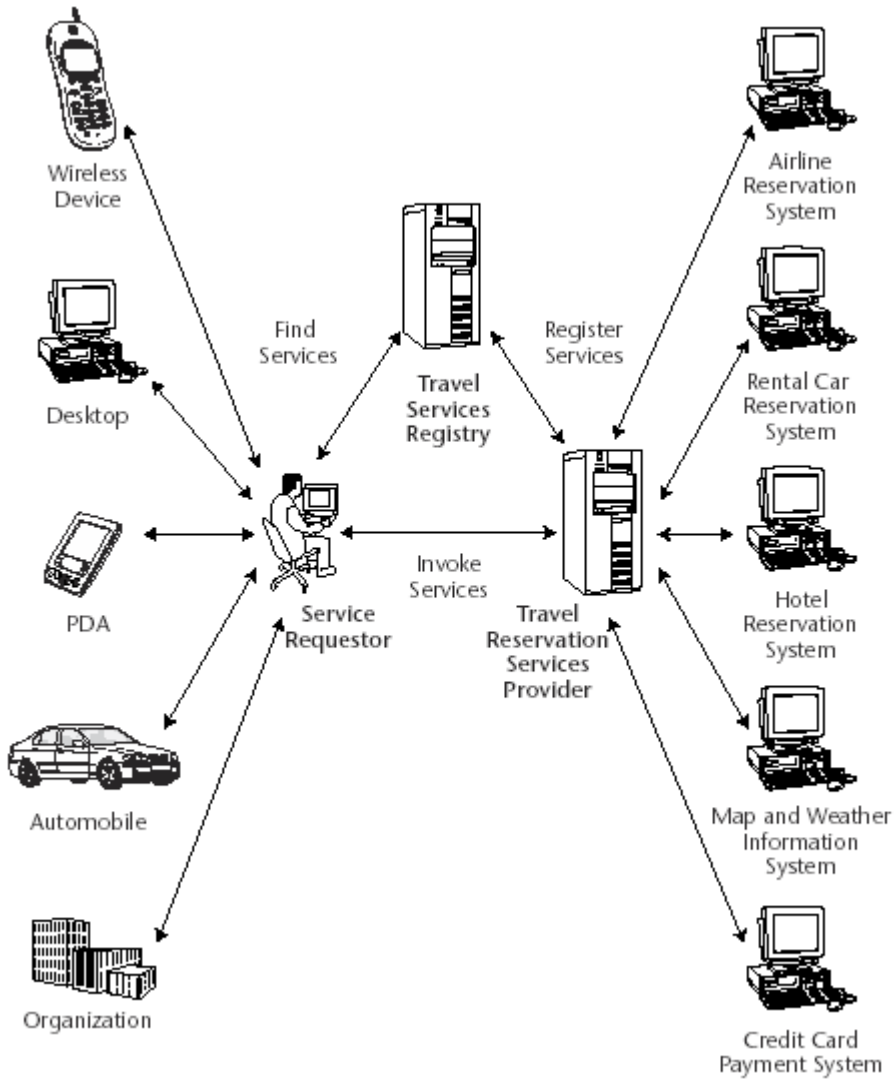


Figure 2: A Simple Web Service

A typical scenario is the following:

1. The travel service provider develops its own web services exposing business proposals secured from various travel companies, such as airlines, car rentals, hotel accommodation, payment, credit cards and more.

2. The service provider registers its services available along with descriptions of using private or public listing. The list of entries store service related information given by the provider of services.

3. The customer discovers web services using a search engine or locating them directly from the list and then invites them to make travel reservations and other functions to the Internet using any platform or other device.

4. In the case of large organizations, business applications using these web services to offer travel services to their employees through the internal corporate network.

The previous example provides a simple scenario of how the business functions of an organization can be presented as web services and invited guests.

1.3: WHY USE WEB SERVICES

Traditionally web applications enable the interaction between the end user and the web page while the web services ,are service-oriented, enable communication from application to application through the internet and easy access to heterogeneous applications and devices. The following are the main technical reasons for using web services instead of web applications:

▪ Web services can be called using XML-RPC. This protocol uses XML to encode its calls and HTTP as a transfer mechanism. It has the benefit that it is not hampered by firewalls.

▪ Web services are platform and language independent since they are based on the exchange of XML messages.

▪ Web services facilitate the implementation of applications using an infrastructure that does not affect scalability.

▪ Web services enable interoperability between heterogeneous applications.

1.4: WEB SERVICES TECHNOLOGY

Web Services are categorized into several layers. Starting from the lowest level, which allows the transfer of data from one machine to another, other higher layers are built on top hiding though the implementation details and ensuring interoperability.

Web Services Technology Stack

Layer Description	Implementation(s)	Other Concerns			
Standard Messaging	Electronic Business XML Initiative (ebXML)	Quality of Service	Management	Security	Service Development
Service Composition	Business Process Execution Service for Web Services (BPEL4WS)				
Service Registry	Universal Description, Discovery and Integration (UDDI) ebXML Registries				
Service Description	Web Services Description Language (WSDL)				
Service Messaging	Simple Object Access Protocol (SOAP)/Extensible Markup Language (XML)				
Service Transport	Hypertext Transfer Protocol (HTTP) Simple Mail Transfer Protocol (SMTP) File Transfer Protocol (FTP)				

Figure 3: Web Service Stack

1.4.1: SERVICE TRANSPORT

The Service transport layer has the responsibility of the data transmission from one service to another. The web services use different protocols to transfer data from service to service, including HTTP, SMTP, FTP, however other protocols can be used as well.

The most common protocol which used by web services is HTTP. It is widely used because it is not blocked by firewalls and so is the standard for interoperable systems.

1.4.2: SERVICE MESSAGING

The Service Messaging layer describes the types of data to be transferred from one service to another. XML is the basic format used for web services. These data formed the basis of XML (tree-like) with the elements and the whole tree form is called document. XML does not require separate description database because data describe themselves and that's because they have special tags that give them a name and a place in the tree-like form.

The SOAP (Simple Object Access Protocol) is a specification that tells the consumer and a service provider in this form and how to read a message written in XML that is used by a service. A SOAP message has three areas: the envelope, the header and the body.

1.4.3: SERVICE DESCRIPTION

The level of description defines three aspects of a web service:

- Methods which a web service exposes.
- The messages received by a web service.

- The protocol, which must be used by the client to call/use the web service.

The Web services use a description language, in particular WSDL (Web Services Description Language), to define a service contract. The service contract is a description of a set of endpoints that sent or receive messages with a specification of how to create an XML message to be sent to the endpoint. The endpoint is a web address that receives messages specially designed according to the requirement stipulated in WSDL. WSDL uses the term port to describe the endpoint that will receive the message and describes the service contract as a collection of ports which the department has made available.

The client uses a service description in two ways. At the time of development of the service (development time), the client can create a strain (stub) that uses the description thereof. The stub is a class which agrees with the description of the service. The client connects to the member at the time of writing (compile time). This is called early binding. The Web services also support the idea of late binding, a bind that is done at the time a service is running between the customer and the service provider. This is done using a dynamic proxy which formed dynamically at execution time of using the WSDL description of the service.

1.4.5: SERVICE REGISTRY

The Web services support the idea of dynamic finding services. A customer uses the service repository to discover services that interest him. The UDDI (Universal Description, Discovery and Integration) is a web service standard that supports a set of services which allow a user of a service to dynamically discover and trace the outline of an internet service. The UDDI stores are themselves web services that expose an API as a set of well-formed SOAP messages. Both a service provider and a consumer can use SOAP and HTTP to publish a service to the UDDI store and receive information for any other interested parties. Public storage services provide information on where you can call the service, the owner of the company that provides the service and any technical information about the web service. The UDDI supports two types of calls:

- The service provider uses UDDI repository to publish information about their services.
- The consumer of a service sends XML messages formatted according to the SOAP service to the store to receive a list of services that fit his criteria.

1.4.6: SERVICE COMPOSITION

Services composition refers to the ability to combine services in a workflow. This is also referred to as "service orchestration" or "choreography services". With services composition one can put in a logical order direct conversations between services in a larger transaction. For example, a transaction that adds a client service bank accounts may well create further accounts to provide information to the customer service. All these applications are driven by a broader entrepreneurial procedural diagram which either succeeds or fails entirely.

There are numerous proposed languages to describe a business procedural diagram. The most famous ones are WSFL (Web Services Flow Languages) and XLANG. Recently IBM, Microsoft and BEA combined the above languages in a specification called BPEL4WS (Business Process Execution Language for Web Services).

1.5: APPLICATIONS SCENARIOS

Web services can be used in many applications, ranging from household tools to video games, and this is why they have been implemented to solve different types of problems. Although web services are wide applicable, more attention has been given to businesses that use it for commercial and inter-integrated business applications.

1.5.1: BUSINESS-TO-BUSINESS

The most popular use of web services is for inter-business transactions. Web services can be used in public networks (on the internet) or in private ones (home/business networks). Companies formed and joined to trusted colleagues in organizations in which members agree to external semantic criteria for communication. Web services are actually a set of standards that provide interoperability in these commercial communities. There are organizations, such as OASIS, that create criteria for e-commerce. The ebXML, a specification for XML messages in e-commerce provided by OASIS with the UN / CEFACT.

The web services do not provide collaborative business procedural capabilities from their own. Some standards like ebXML provide a basis for complex interactions with messages between organizations, facilitating their trading partners to engage in open and complex business processes.

With the convergence of Internet, Web services and standards for the transmission of messages, organizations can direct a computer job with a defined clearly.

1.5.2: ENTERPRISE APPLICATION INTEGRATION

The most common usage of web services is in integrated business applications. The business applications integrated are a solution to the problem of having many applications that communicate with each other. Substantially modify the protocols and data formats to be operable communication and understanding between applications. So basically this is the problem solving and web services.

The vendors have created products integrated business applications to solve the problem of communication between applications in an enterprise. These products are designed to solve problems 3:

- The level communication cable.
- The conversion of the data can be understood in different kinds of applications.
- Determining the route messages between systems.

Typical products business applications integrated provide tools that help you customize a protocol to another or provide a standard for interoperability. The typical solution of the business applications integrated, shown in the figure below, followed by the "hub-and-spoke" model for integration applications. According to this, all applications connected to a central business application integrated and any application that wants to communicate with another must first send the message to the central business applications integrated, which transforms and routes the message to the recipient by the sender.

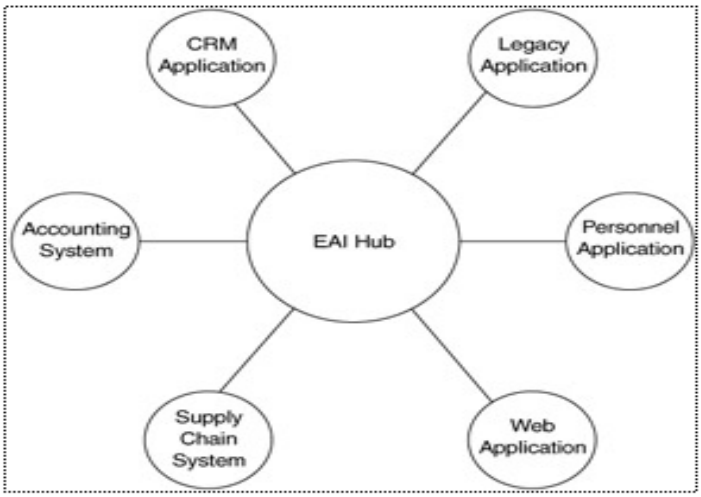


Figure 4: hub-and-spoke topology of enterprise application integration

Some products allow the sender and receiver use different protocols. For example, for the buying a customer relationship management system (CRM), an organization needs to install data for this system of inherited COBOL applications that are in the main frame. The CRM application can use

a fixed intermediate protocol such as IBM's MQSeries, to communicate with the central point of business applications integrated, or may use a different protocol such as HTTP. The central point of the business applications integrated will accept the message over different protocols will transform into an understandable message from COBOL structure and lead the message to the host system. All viable vendors of the business applications integrated add to their products support services of web services. This will allow products of the business applications integrated to route the messages to other web services protocols and data formats.

But the solutions of the business applications integrated have some drawbacks. They are expensive and several externalize business rules outside of the application. Typical products of the business applications integrated with specific languages to describe the transformation rules and routing data. The rules for transforming data using data tables and require special knowledge on the data conversion. Some products of the business applications integrated, if not managed properly, can save much of the business rules for an application. The hub-and-spoke model of communication makes the integration and management of the business applications integrated solution easier, but carries a single point of failure that leaves many vulnerable systems.

The web services solve some of these problems. The business logic is externalized. The logic to transform the data into a format understood by the service is more relevant to the service. Eventually the web services do not follow the model of hub-and-spoke. Each connection from the consumer of web services to these providers is a direct connection without interference third. So the problems just focus on individual departments, allowing others to operate separately. Moreover, instead of using the suggestions for translations of protocols and data formats of the business applications integrated, systems communicate directly with each other through web services, as shown below. Instead of having a single point of failure by using the topology hub-and-spoke, the web services form a semi-lattice topology for communication applications.

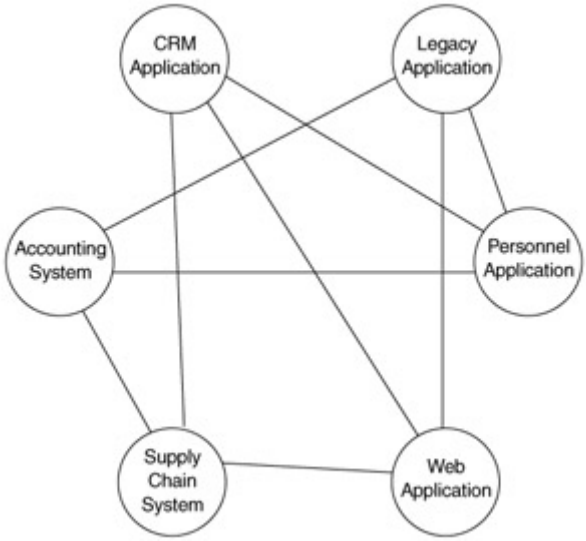


Figure 5: Web Service topology integration of business applications

1.6: IMPLEMENTATION SCENARIOS

A web service can be deployed on many different platforms, and can be followed by many models of the implementation. The implementation model that uses depends on the problem to be solved and the available platforms to create the model. The model for the integration of a web service can be categorized into 4 types: simple service, composite service, interim service and a bus service.

1.6.1: SIMPLE SERVICE

A simple service, as shown in the figure below is a servlet that has access to a database or other source directly. If a company, for example, has a session bean without statements which handles the validity of credit cards, the JAX-RPC can be used to provide a web service interface.

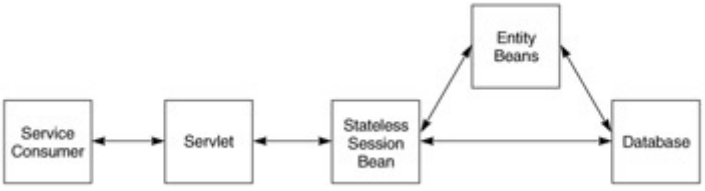


Figure 6: Simple web service scenario

1.6.2: COMPOSITE SERVICE

A composite service integrates other simple or complex applications. For example, an “order-entry” service would be a complex service that uses a credit card authorization service, a service client and a service “order-entry”. The combination of these three functions into one composite gives extra functionality to the consumer and also the ability to use each function separately or combining them through an order-entry service.

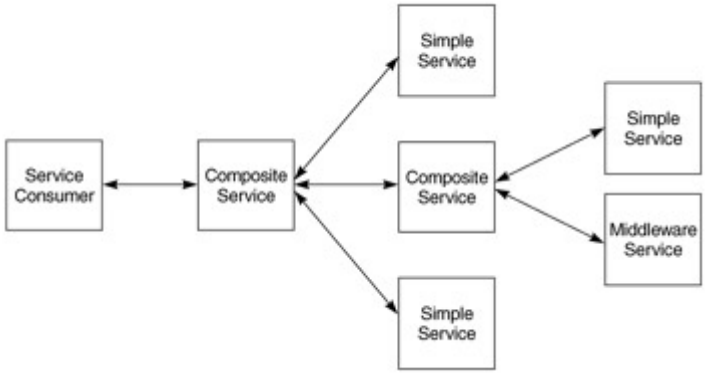


Figure 7: Composite web service scenario

1.6.3: INTERIM SERVICE

A web service that places requests in a middle aisle is an example implementation of interim service as shown below. Many organizations have already installed interim systems oriented messages. Such a system provides an indirect connection between the sender and the receiver of the message. The sender puts the message in the queue, which sends messages to an endpoint, whose exact location is not known implementation of the sender.

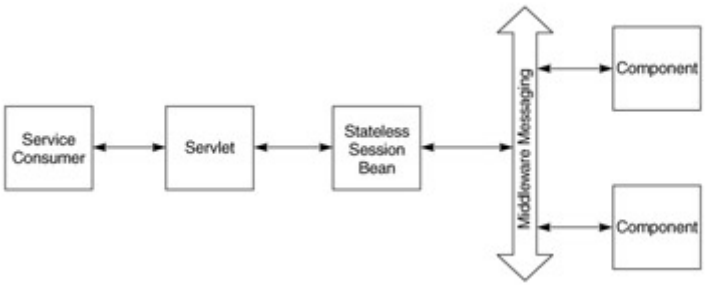


Figure 8: Interim web service scenario

1.6.4: BUS SERVICE

Corridor services as shown in the figure below are similar to the solution of the business applications integrated for the completion business applications. They allow web services to communicate through the bus.

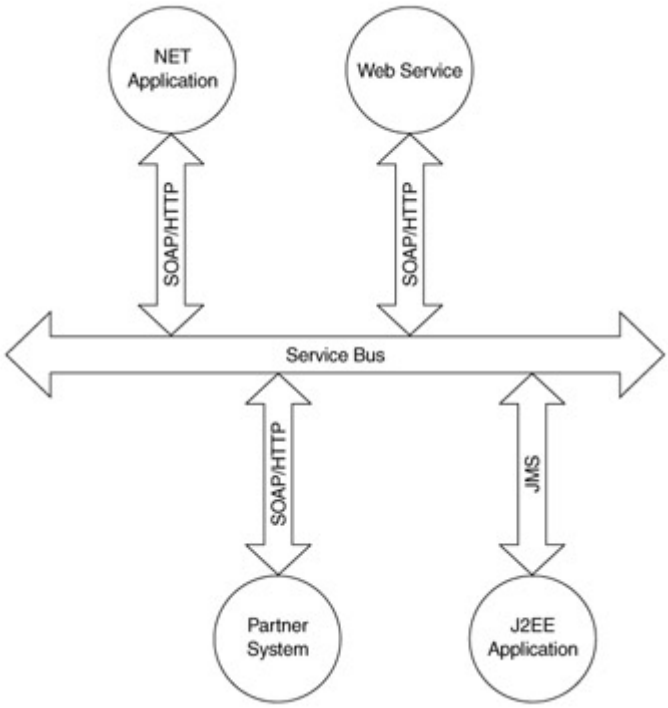


Figure 9: Bus web service scenario

For example, a banking application that consists of a graphical user interface (GUI) through which the user can make banking transactions from an accounting department that processes deposits. When the system starts working, the accounting department records the interest on deposits with the help of a service corridor. Suppose the GUI accepts a deposit request from a user. The GUI instead communicates directly with the accounting department and places the request in the bus services. Since the accounting department has already registered its interest for applications deposits, bus forwards the request to the accounting department by the user. The accounting department receives the application filing the aisle, processes it and returns the results to the service bus. Then, through the accounting service, the bus promotes the results in graphical user interface presentation.

This method of routing messages between web services is multifaceted. The accounting department takes case of placing a message in the bus services, and other services can also register their interest. If a log in service or another service management of customer transactions and interested while those for deposits of requests from customers who appeared in the bus could also those to register their interest in these transactions. This is a multi-cast communication method by which a user can send messages to

multiple services simultaneously. The user does not know what other additional services are involved in the application. Management software directs service bus to route these messages. The bus also provides many other solutions to the integrated business applications such as transformation of applications and format into another. In the above scenarios there are specific benefits of using web services.

CHAPTER 2

2. SOA - BASIC MODEL OPERATION OF WEB SERVICES

Web services are built based on the idea of service-oriented architecture (SOA). SOA is the latest evolution of distributed systems calculation which enables software components to be presented as services.

Each SOA has three roles and relationships that define the providers and users of web services. These are: the user of a service (service requestor), the provider of a service (service provider) and the list of available services (service registry).

- The provider is responsible for creating the service description for the publication of this description in one or more directories of services and for the receiving web services from one or more users. A service provider can therefore be a company which can host a network of web services.

- The service user is responsible for identifying a service description which is published in one or more directories of services and is also responsible for using these descriptions to connect or call web services hosted by different providers.

- The services registry is responsible for advertising the descriptions of web services that are published to it from various providers and allows users to search collections descriptions of services listed. The role of the registry is simple: to make the right match between the applicant and a service provider. Once the matching is done, it interacts directly between the user and the provider.

SOA also includes three functions: publish, find and bind. These functions define the contracts between SOA roles:

- The publication is a record of a transaction or service advertising. It acts as a contract between the directory service and the provider. When a provider publishes a description of the web service in a registry of services, then this registry, advertise the details of web services to a community of users. The exact details of how the publication, depends on the specific

implementation of the registry. In some scenarios for immediate publication, the network plays the role of directory services, making the simple activity of publishing a description of the transport service in directory structure of web server applications. Other implementations of registry, such as UDDI, define a complex implementation process of publishing.

- The process of finding is the opposite to publication. The finding process is a contract between the applicant of a service and directory of services. In the process of finding the user declares some features, such as the type of service, quality, parameters and others. The registry of services matches the criteria specified by the applicant with the published web service descriptions. The result of the operation is finding a list of web service descriptions that meet the above criteria. Of course, the process of finding varies depending on the inventory service. On the one hand, there are simple registries of services that provide procedures to find anything more than a simple request parameters without HTTP GET. This of course means that the process of finding will always return all the services that are registered in the registry and it is up to the user to find which of these satisfy his conditions. On the other hand, of course, there are more complex registries, such as UDDI, offering excellent potential finding.

- The binding process embodies the client-server model between the applicant and the service provider. The binding process can be quite complex and dynamic, such as dynamic creation of an attorney client-based on description of the service and is used to call the service. It can also be a static process when someone can write code for the way which the client application will call the web service.

The key to SOA is the description of the service. It is the one published by the provider to the registry of services. The description of the service is recovered by the applicant as a result of a process of searching. The description of the service also provides to the applicant the required information in order to connect/call/exploit the web service. The description of the service, finally, determines what information is returned to the applicant as a result of calling the web service.

CHAPTER 3

3. WSDL - WEB SERVICES DESCRIPTION LANGUAGE

Web Services Description Language (for short WSDL) is a specification which is used for describing web services using XML grammar. WSDL describes four important pieces of data:

- Information of all publicly available functions.
- Information of data types for request and response messages.
- Connection information about the transport protocols.
- Addresses information to identify the specific service.

Using WSDL one client can locate a web service and call any public functions.

3.1 WSDL SPECIFICATION

WSDL is an XML grammar for describing web services. The structure of a WSDL document is shown below:

```
<definitions>
    <import>*
    <types>
        <schema></schema>*
    </types>
    <message>*
        <part></part>*
    </message>
    <PortType>*
        <operation>*
            <input></input>
            <output></output>
            <fault></fault>*
        </operation>
    </PortType>

    <binding>*
        <operation>*
            <input></input>
            <output></output>
        </operation>
    </binding>
    <service>*
        <port></port>*
    </service>
</definitions>
```

Figure 10: WSDL document structure

WSDL specification is divided into 6 main elements:

- **Definitions:** This element is mandatory and must be the root of the xml structure for each WSDL. It specifies the name of the Web service, declares multiple namespaces used in the rest of the text and contains all the other elements of the service.

- **Types:** This element describes all the data types used between the client and the server. WSDL does necessarily not follow particular reporting system types, but uses the W3C XML schema specification as the default choice. If the service uses only XML structured shape in simple formulas such as strings and integers, the element types is not necessary.

- **Message:** This element describes a one way message whether it is used as a request or a response. It specifies the name of the message and contains (zero or more) 'part' elements, which refer to message parameters or return values.

- **PortType:** The PortType element combines multiple ‘message’ elements to compose a simple function. For example, one may combine using PortType a message request and a message response to a unique function request-response which is used most often in the SOAP services. It should be noted that PortType often determines complex functions.

- **Binding:** The binding element describes how to implement this service over the bus.

- **Service:** To service element defines the address for the service calling and contains the URL of the service.

In addition to these six basic elements, the WSDL also defines the following useful information:

- **Documentation:** The documentation element is used to provide a human-friendly description of the service and can be included in any WSDL element.

- **Import:** The import element is used to import WSDL files.

3.2 BASIC EXAMPLE IN WSDL (HelloService.wsdl)

In order to better clarify the above concepts we'll describe them in the following example. The 'HelloService.wsdl' document describes a service, 'HelloService'. As it is depicted in the following figure, the service provides a public method, 'sayHello'. This method expects as parameter a string type parameter and returns a simple greeting, again type string. For example, if we pass the parameter 'world', then the service will return «Hello, World!".

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloService"
   targetNamespace="http://www.ecerami.com/wsdl/HelloService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.ecerami.com/wsdl/HelloService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <message name="SayHelloRequest">
      <part name="firstName" type="xsd:string"/>
   </message>

   <message name="SayHelloResponse">
      <part name="greeting" type="xsd:string"/>
   </message>

  <portType name="Hello_PortType">
     <operation name="sayHello">
        <input message="tns:SayHelloRequest"/>
        <output message="tns:SayHelloResponse"/>
     </operation>
  </portType>

  <binding name="Hello_Binding" type="tns:Hello_PortType">
     <soap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http"/>
     <operation name="sayHello">
        <soap:operation soapAction="sayHello"/>
        <input>
           <soap:body
              encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
              namespace="urn:examples:helloservice"
              use="encoded"/>
        </input>
        <output>
           <soap:body
              encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
              namespace="urn:examples:helloservice"
              use="encoded"/>
        </output>
     </operation>
```

Figure 11: HelloService.wsdl

3.2.1 Explanation the data of the example

a) Definitions

The element 'definitions' specifies that this WSDL the 'HelloService'.It is actually an identity of the WSDL. It also includes various namespaces that will be used throughout the text.

```
<definitions name="HelloService"
   targetNamespace="http://www.ecerami.com/wsdl/HelloService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.ecerami.com/wsdl/HelloService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

The use of namespaces is important to distinguish elements and enables the document WSDL to refer to external standards, including WSDL, SOAP and XML Schema specification.

The element 'definitions' also declares a property ''targetNamespace'. This property is a convention of XML Schema that enables the WSDL document to refer to itself. In this example we set as 'targetNamespace' in 'http://www.ecerami.com/wsdl/HelloService.wsdl'. We should note, however, that the specification of the namespace does not require that there is indeed a WSDL document at this location. What is important is to set a value that is unique and different from other namespaces.

Finally, the element 'definitions' declares one XML-specific namespace, the 'xmlns = http://schemas.xmlsoap.org/wsdl/'. Each item that does not have a prefix such as 'message' and 'portType', are considered as elements of this namespace.

b) Message

There are two 'message' elements mentioned in this example. The first, (“SayHelloRequest”), represents a request message, the second one, (“SayHelloResponse”), represents a response message.

```
<message name="SayHelloRequest">

      <part name="firstName" type="xsd:string"/>

</message>

<message name="SayHelloResponse">

      <part name="greeting" type="xsd:string"/>

</message>
```

Each message contains a unique element 'element'. For the application, this element specifies the parameters passed to a function. In this case we declare a single parameter, the 'firstName'. Regarding the response, 'part' defines the return values of the function. In this case we have only one, the return variable 'greeting'.

The attribute 'type' of the element 'part' defines a type conforming to the XML Schema. The value of this attribute must be given as XML Schema QName, i.e. has a prefix that refers to a namespace. For example, the property 'type' element 'firstName' has value 'xsd: string'. The 'xsd' prefix refers to the namespace of the XML schema defined in the namespaces of element 'definitions'.

If a function wait multiple arguments and return multiple values, then we can define for each one of them an element of 'part'.

c) portType

The element 'portType' defines a simple operation called "sayHello". This function includes a single input message (SayHelloRequest) and a single output message (SayHelloResponse):

```
<portType name="Hello_PortType">

   <operation name="sayHello">

          <input message="tns:SayHelloRequest"/>

          <output message="tns:SayHelloResponse"/>

   </operation>

</portType>
```

Similarly to the property 'type' mentioned previously, the 'message' property should be defined as XML Schema QName, ie to refer to a namespace. For example, the element 'input' contains an attribute 'message' with value "tns: SayHelloRequest". The prefix 'tns' refers to the 'targetNamespace' defined within the element 'definitions'.

WSDL supports four different modes:

- **One-Way:** The service receives the message. So the element 'operation' is a unique element 'input'.

- **Request - Response:** The service receives a request message and sends a reply message. So the element 'operation' has an element 'input' followed by an element 'output'.

- **Solicit Response:** The service sends a message and then receives a response. So the element 'operation' has an element 'output' followed by an element of 'input'.

- **Notification:** The service sends a message. Thus, the element operation' is a unique element 'output'.

These four functions are shown below and schematically. The model of request-response, however, is the most widely used in SOAP services.

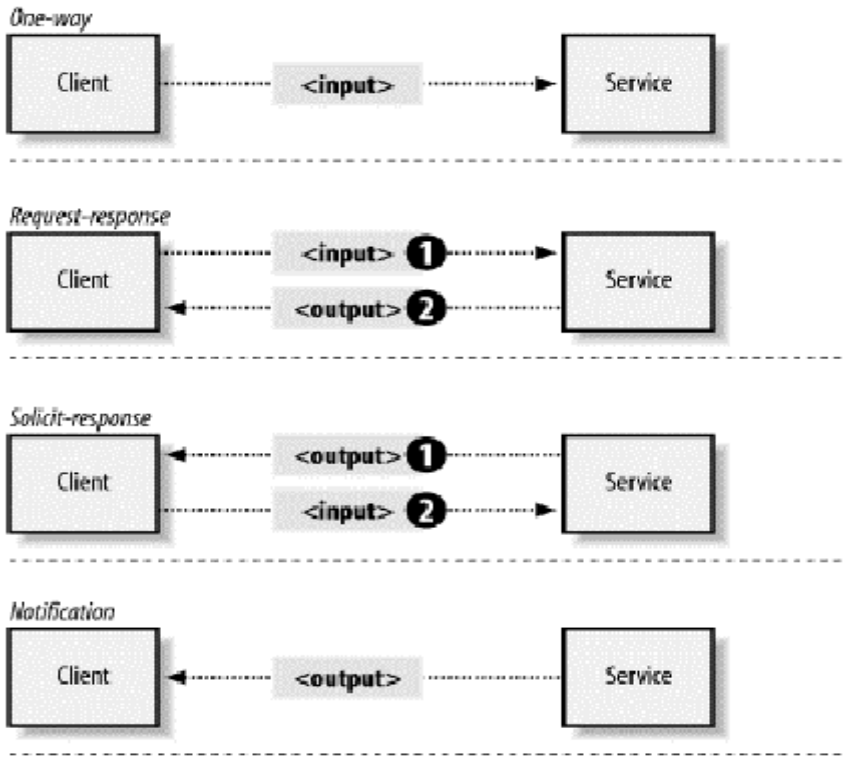


Figure 12: The four standard modes supported by WSDL

d) binding

The element 'binding' provides specific details on how a function defined by the element 'portType' will be transferred over the bus. Connections can be made using various protocols, including HTTP GET, HTTP POST, or SOAP. In fact someone may define many connections (bindings) for each element 'portType'.

Each element 'binding' determines the properties 'name' and 'type'.

```
<binding name="Hello_Binding" type="tns:Hello_PortType">
```

The attribute 'type' refers to the element 'portType' name 'Hello_PortType' that is set above the WSDL document. In this case, the element 'binding' has as a (‘type’) value 'tns: Hello_PortType' which refers to the targetNamespace. Essentially, the element 'binding' provides specific details about how the 'sayHello' function will be transferred through the internet.

CHAPTER 4

4. BPEL – BUSINESS PROCESS EXECUTION LANGUAGE

4.1 WHAT IS A WORKFLOW?

Web services are self-contained applications that are described by WSDL language, discovered and published in UDDI and communicate through the SOAP protocol. It is easier for users to connect different pieces of work across different business units in a platform independent programming language and manner.

However none of these standards specify a description for the correlation of web services together. Thus, web services remain isolated and opaque. To break this isolation requires connecting web services and determine how various collections of web services can be combined with each other and create a more complex function, a workflow (business process).

A workflow (business process) defines a model and a grammar for describing the behavior of a workflow. It specifies the sequence of operations on a collection of web services, the data exchanged between them, what partners are involved and how they are involved in the process. The interaction between each partner occurs through web service interfaces and the structure of the relationship between the partner and a process embedded in a partner link. A BPEL process defines how the various interactions between the partners and services are arranged to achieve a purpose. It also introduces mechanisms for handling and processing of errors and finally introduces a mechanism for retrieval when an exception occurs or when the partner asked undo actions.

BPEL combines Web Services Flow Language IBM and Microsoft's XLANG and creates a new standard based on two previous.

As a language that implements executable processes, BPEL essentially must define a new web service composed of a set of existing web services. So basically BPEL is the language that can implement this composition. The composition is described by a set of WSDL portTypes, as any other web service, called flow (process).

The following figure shows a flow:

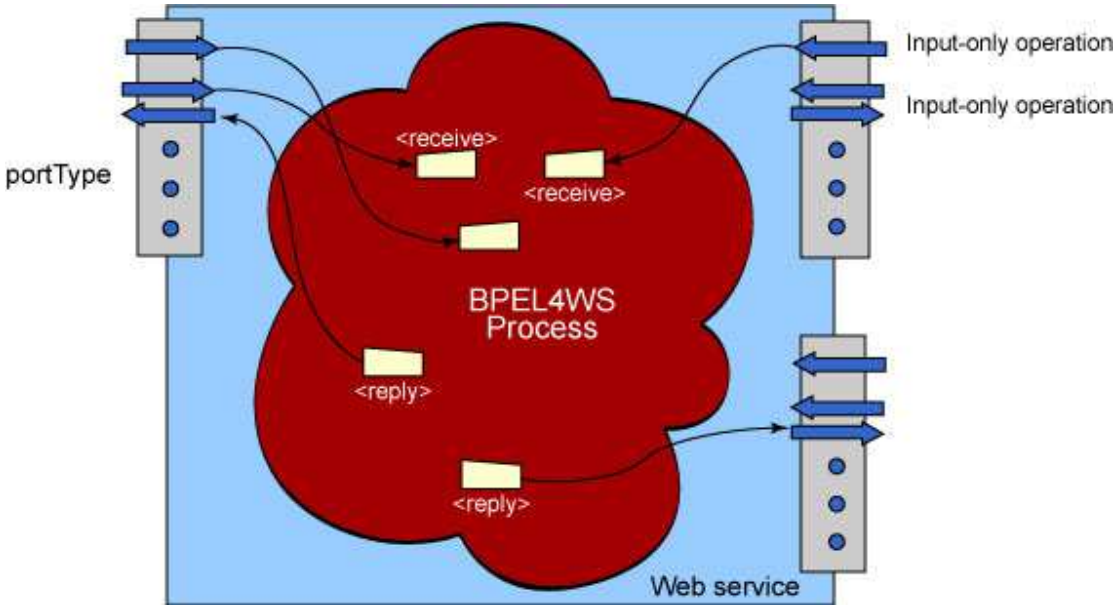


Figure 13: Picture of a Web service that is implemented as a BPEL flow

4.2 IMPLEMENTATION OF WORKFLOW

What is in the cloud? Unlike traditional language implementation of a WSDL service, each one portType operation does not correspond to a separate piece of logic in BPEL. On the contrary, the whole set of PortTypes service is implemented by a unique BPEL process. Thus, specific points of entry, which enable users to call the external operations of the procedure indicated in the flow. At these points entering or leaving various messages of their respective functions of input-only entry or input-output. The single-output functions (notification) and input-output (solicit-response) are not required and are not supported.

4.2.1 ACTIVITIES

A BPEL process is basically a flow chart. Each step in the process is called action (activity). There are some basic steps, such as:

- <invoke>: calling a function
- <receive>: waiting for a message to be received
- <reply>: send the reply from an input-output function
- <wait>: waiting for a time
- <assign>: copy data from one point to another
- <throw>: indicates that a mistake has occurred
- <terminate>: ending the entire flow
- <empty>: do nothing

These basic steps can be combined into more complex algorithms to implement the entire flow. So someone can create

- a sequence of web services to be performed in sequence with the help of energy <invoke>
- branching flows with the help of energy <switch>
- loops with <while>
- executing a flow selection among many using the <pick>
- and finally a parallel workflow with the help of <flow>. In parallel workflow can define a job when you start the next with the help of links.

4.2.2: PARTNERS: INTERACTING WITH OTHERS

The Language of composing a set of web services into one, BPEL contains calls to other services or receives calls from customers who are the users who send messages in the figure above.

The call to other services is done in terms of <invoke> and accept calls using the <receive>. BPEL calls the service with which it interacts partners.

Thus, one partner can be one of the following three:

- services which call flow
- services which call flow
- or services that call the flow and which the call flow (whichever may occur first)

4.2.3: SERVICE LINK TYPE

To describe the relationship of the third type of partner should be seen from either side of the service call flow or side flow calling services. To avoid this, with the help of service link types, we can describe the relationship between the two in a third person. A service link type describes a set of roles, each role corresponding to a set of portTypes. The central idea is when there are two interacting services, then the service link type determines how they interact, particularly what services are offered each side.

BPEL uses service link types to determine partners. To define a partner one should give it a name, and then determine what service link type mentioned and finally determine the role to be played in BPEL flow and the role which will have the web service.

4.2.4: HOW TO INDICATE THE PARTNER IN WORKFLOW

What one partner do during execution? A partner is simply a written reference to a service (web service). The word 'written' refers to the

description of two elements for a partner, a "<partnerLinkType> and a " <role> " for indication which service can be called and in what property.

4.2.5: FACING THE PROBLEMS AND MISTAKES IN THE EXECUTION

People, who design workflows, must have some policy when handling errors that occur during the execution of the workflow. BPEL language has built some steps to throw and catch exceptions. This is done through the "throw" and "catch" elements.

Furthermore, BPEL supports the notion of 'remediation' (compensation), which helps the designer to implement alternative operations for some non-reversible reactions. For example, let's consider a travel ticket booking process. If a reservation is active, which is an operation which cannot be reversed, in order to make cancellations some other different activities must be driven. These actions are called 'alternative energies' for the energy of the reservation.

Handling errors and alternative energies are supported by the concept of 'scope'. This element describes the content of each operation. It includes handling errors, manipulations reparations, definitions of variables that receive data definitions, 'partner links' definitions and correlations definitions (correlation sets).

4.2.6: LIFETIME OF WORKFLOWS

When web services are combined into workflows then the lifetime is instantaneous. This means that every customer service interacts with one instance of the workflow.

When web services are combined into workflows then the lifetime is instantaneous. This means that every customer service interacts with one instance of the workflow.

How to create those shots yet each workflow? Those are implicitly created by the arrival of messages for workflow. Those snapshots are not specified by a 'instanceID', but some key-value pairs included in the messages. For example, if a workflow describes the execution of an order, the order number can be the key to identify the specific snapshot. So when you send a message to start a workflow and there is no instance that matches the key-value that carries the message, then a new snapshot is created that automatically identifies the value of the message key.

Messages that are sent to a workflow either create a new snapshot or retrieve one if the appropriate existing snapshots are found, which wait to receive a message to another point in the flow, except for the starting point.

The process of creating a new instance or to identify an appropriate snapshot to deliver the message is called 'correlation messages'.

4.2.7: WORKFLOW STARTING

A workflow can be started when you sent a message. To receive this message the activity <receive> should be the first one, which includes the 'createInstance' with value 'true'.

4.3: A SIMPLE EXAMPLE OF A WORKFLOW – GET A LOAN

We will present a simple example to understand how different actions can be created and combined with BPEL. The example refers to the execution of the procedure used to obtain a loan. We will present the key actions that will be used in the workflow, together with the WSDL descriptions of web services that are related and will be used in the process. As you create the workflow it will also explain and use of associates to call the service, use of variables that receive the messages exchanged and then use the actions (activities) with which to communicate with the outside world, which are <receive>, <invoke>, <reply>.

More specifically, in our example the client sends a request for a loan. The application process and the customer get a response if the loan is approved or not. The processing of the customer request including being sent is an organization that provides the service of processing the loan request and sends back the response. The client-side only sees that the process will receive the request and will return the answer. The diagram below shows the part of the client how can see the external workflow, which occurs as the cloud shape.

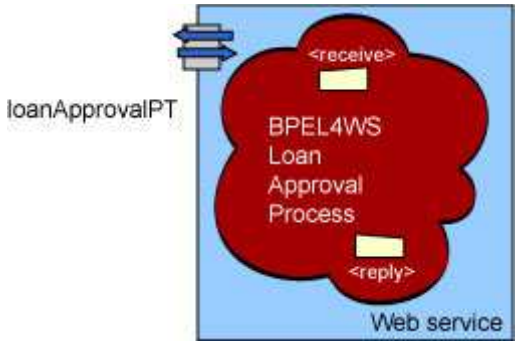


Figure 14: Exterior view of the workflow for a loan getting

4.3.1: STRUCTURE OF THE WORKFLOW

The workflow will start waiting for receiving a message, then the service organization will be called to process the request and then the response will be send back to the client. These three steps are described in BPEL with the actions <receive>, <invoke>, <reply>. But further clarification is needed to describe how these effects relate to each other to know the process of how to perform them. Related actions described in BPEL with some actions that impose restrictions on how to invite the actions contained therein. In our example we want to make three actions sequentially, one after the other. This sequence can be achieved with the help of action <sequence>, which contained the first <receive> to the message, then the <invoke> to call the service and finally the <reply> to send the answer to the user. So the cloud in the figure below will contain the workflow, which includes action <sequence> other three operations and may invite the external web service body to process the request

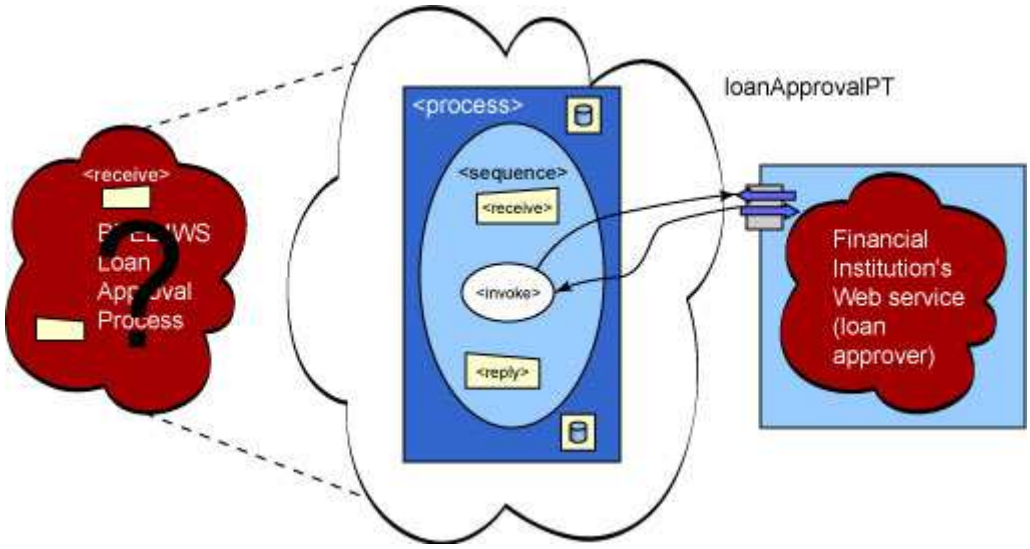


Figure 15: What's inside the Cloud? Internal view of the workflow for a loan

4.3.2: DESCRIPTION OF WEB SERVICES THAT WILL COOPERATE FOR YOUR WORKFLOW

Before we proceed to a more detailed description of the workflow, we must first give descriptions of what will be involved in the process and the messages that will be exchanged.

The composition of the workflow is based much on WSDL descriptions of web services that participate in order to be able to refer to the messages they receive and dispatch methods (operations) that can be called and what interfaces (PortTypes) they belong. In our example, we will need a description

of the web service financial institution that processes the request to create the workflow. Suppose one wants to make request loans and provide full name and the amount he wishes. These parameters are incorporated into a message that is defined in the description of the web service of the organization (the creditInformationMessage) and returns an integer message type named loanRequestErrorMessage, as shown below:

```
      <definitions
targetNamespace="http://tempuri.org/services/loandefinitions"

xmlns:tns="http://tempuri.org/services/loandefinitions"
                    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                    xmlns="http://schemas.xmlsoap.org/wsdl/">

        <message name="creditInformationMessage">
           <part name="firstName" type="xsd:string"/>
           <part name="name" type="xsd:string"/>
           <part name="amount" type="xsd:integer"/>
        </message>

        <message name="loanRequestErrorMessage">
           <part name="errorCode" type="xsd:integer"/>
         </message>

       </definitions>
```

Figure 16: Loan Definitions WSDL (loandefinitions.wsdl)

Suppose that the financial institution provides a web service for loan approval. The following description of this web service, present the only method (operation), named "approve", which decides whether to approve the loan or not. The operation gets information of the client as input and returns the response to a message of approval or denial, which was described in the above description.

```
      <definitions
targetNamespace="http://tempuri.org/services/loanapprover"

xmlns:tns="http://tempuri.org/services/loanapprover"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:loandef="http://tempuri.org/services/loandefinitions"
                              xmlns="http://schemas.xmlsoap.org/wsdl/">

        <import
namespace="http://tempuri.org/services/loandefinitions"
                  location="http://localhost:8080/bpws-
samples/loanapproval/loandefinitions.wsdl"/>

        <message name="approvalMessage">
          <part name="accept" type="xsd:string"/>
        </message>

        <portType name="loanApprovalPT">
          <operation name-"approve">
            <input message="loandef:creditInformationMessage"/>
            <output message="tns:approvalMessage"/>
            <fault name="loanProcessFault"
                   message="loandef:loanRequestErrorMessage"/>
          </operation>
        </portType>

      <binding ...> ... </binding>
      <service name="LoanApprover">....</service>
      </definitions>
```

Figure 17: Loan Approver WSDL (loanapprover.wsdl)

Moreover, we must define the elements partnerLinkTypes, which define the roles we can call the service. The element partnerLinkType specifies the two roles and refers to assigned PortTypes (interfaces) for services related to each other. Do not forget that the workflow is a new service as well. Therefore, to connect your workflow through a web service is one partner link between the two services. In our example, the partnerLinkType will be used to connect the customer with the workflow, as well as the process by web service of processing the loan. We say two roles, because the workflow and web service processing the loan provide the same

"approver" portType (interface) and none need to use another portType, ultimately having only one role. The description of partnerLinkTypes shown below:

```
      <definitions
            targetNamespace="http://loans.org/wsdl/loan-approval"
            xmlns="http://schemas.xmlsoap.org/wsdl/"

xmlns:slnk="http://schemas.xmlsoap.org/ws/2002/06/service-link/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:lns="http://loans.org/wsdl/loan-approval"
            xmlns:apns="http://tempuri.org/services/loanapprover">
        <import namespace="http://tempuri.org/services/loanapprover"
                location="http://localhost:8080/bpws-
samples/loanapproval/loanapprover.wsdl"/>
        <import
namespace="http://tempuri.org/services/loandefinitions"
                location="http://localhost:8080/bpws-
samples/loanapproval/loandefinitions.wsdl"/>
        <slnk:serviceLinkType name="loanApprovalLinkType">
          <slnk:role name="approver">
            <portType name="apns:loanApprovalPT"/>
          </slnk:role>
        </slnk:serviceLinkType>
        <service name="loanapprovalServiceBP"/>
      </definitions>
```

Figure 18: Loan Approval WSDL (loan-approval.wsdl)

4.3.3: CREATING THE WORKFLOW

All preconditions are now available and we can proceed to build the workflow. We start the description with the element <process> and mention the namespaces that will need to refer to the corresponding WSDL which describe the service and the messages that we will use. The namespaces that will be needed is the target namespace of the web service loan approver (http://.../loanapprover), the target namespace of partnerLinkTypes (http://.../loan-approval) and the target namespace of message (http://.../loandefinitions). The workflow is now ready to use the web service as accessory.

```
      <process name="loanApprovalProcess"

targetNamespace="http://acme.com/simpleloanprocessing"

xmlns="http://schemas.xmlsoap.org/ws/2002/07/business-process/"
                  xmlns:lns="http://loans.org/wsdl/loan-approval"
```

```
xmlns:loandef="http://tempuri.org/services/loandefinitions"

xmlns:apns="http://tempuri.org/services/loanapprover">
```

Figure 19: Workflow namespaces

The next step is to determine the partners in the workflow; the names of the partners and how partnerLinkType are identified in loan-approval.wsdl description. For example, our partners are the customer and the organization providing the loan. The property myRole / partnerRole of <partner> element specifies how partners and the workflow will interact given the partnerLinkType. The property myRole refers to the role partnerLinkType will play in the workflow, the property partnerRole refers to the role that will be played by each partner. These are determined by the definition of partner shown below.

The workflow offers loanApprovalPT interface to the customer and the financial institution offers this interface in workflow. This relationship is shown in the above two figures.

```
<partners>
    <partner name="customer"
             serviceLinkType="lns:loanApproveLinkType"
             myRole="approver"/>
    <partner name="approver"
             serviceLinkType="lns:loanApprovalLinkType"
             partnerRole="approver"/>
</partners>
```

Figure 20: Definition of partners

After setting the partners, we're almost ready to add the actions in the workflow. Let's see what we should do. For requesting the customer a loan, a message must sent to the workflow, the workflow will then ask the agency if request is accepted or not, and will respond to the client by sending another message of acceptance or denial of the application. How is this done in BPEL? First you need to locate the message that will come from the client to a container (variable) which can access the workflow. In BPEL, the data is written and accessed by container data, which can keep snapshots of specific message types which described in WSDL.

From the definition of customer partner and loanApprovalPT, we can understand that the client will send a message type of creditInformation Message and will receive a response type of approvalMessage. So formed and all the receptors (variables).

```
<variables>
    <variable                                    name="request"
messageType="loandef:CreditInformationMessage"/>
    <variable                                name="approvalInfo"
messageType="apns:approvalMessage"/>
</variables>
```

Figure 21: Definition of variables

4.3.4: INTERACTING WITH THE WORKFLOW

The workflow can contain only one action, in this case <sequence>. In <sequence> an action <receive> can be added which expects to receive from the client the message and place it in the appropriate container. The definition of action <receive> must include the name of the partner who will send the message, along with the name of the portType and operation that wants to send the message the partner. According to this information, when the

workflow receives the message, it will try to find a <receive> action that waits for a message with the same partner-portType-operation and delivers the message in this action. To avoid confusion, the BPEL specification declares that there should be no more than one <receive> actions that correspond to the same partner-portType-operation, which are not waiting simultaneously to receive the same message. The action <receive> then places the message in the particular receptor and completed. So we'll start with the action <sequence> and add action <receive> therein.

```
<sequence>
     <receive name="receive1" partner="customer"
                  portType="apns:loanApprovalPT"
                  operation="approve" variable="request"
                  createInstance="yes">
     </receive>
```

Figure 22: Receive action in workflow

The next step is to ask the web service financial institution if it accepts the request or not. This is achieved by making a call to the web service with action <invoke>. When you run this action, the web service will be called by sending the message to the input jack (inputVariable) of web service and storing the response to the output receptor (outputVariable) and completed. We can observe that the call operation "approve" will make the partner "approver".

```
<invoke name="invokeapprover"
                  partner="approver"
                  portType="apns:loanApprovalPT"
                  operation="approve"
                  inputContainer="request"
                  outputContainer="approvalInfo">
     </invoke>
```

Figure 23: Invoke action in workflow

To send the workflow, the answer to the customer uses the action <reply>. Each <reply> should be assigned to one <receive>. The partner-portType-Operation' which contains <reply> should be identified with a triplet <receive> action. The message will be sent back to the client will contain the response which sent the financial institution incorporated in output container (approvalInfo) of action <invoke>. After completion of <reply> the workflows ends closing tags <sequence> and <process>.

```
      <reply              name="reply"              partner="customer"
portType="apns:loanApprovalPT"
                    operation="approve" container="approvalInfo">
          </reply>
        </sequence>
      </process>
```

Figure 24: Reply action in workflow

4.3.5: OVERALL VIEW OF THE WORKFLOW FOR LOAN APPROVAL

After the workflow is created, it will wait for a message to be executed. The action <receive> contains the property 'createInstance' with value 'true'. This suggests a starting point of the workflow. The figure below shows how the workflow runs.

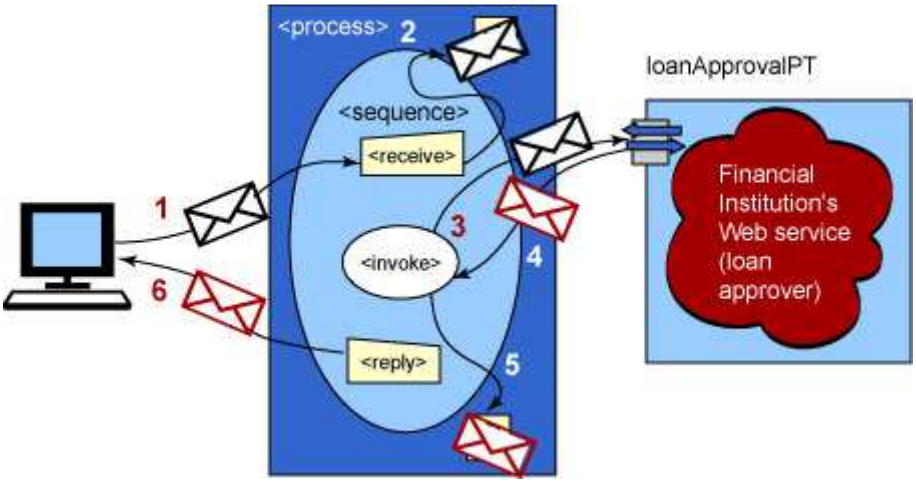


Figure 25: Execution the workflow for loan approval

The numbers in the figure above show the order in which the steps of the process are executed. Folder 1 contains loan application and folder 6 contains the answer.

When the client sends a message to the workflow with this triplet partner-PortType-Operation then a new instance is created and executed. In our example, the process proceeded to <sequence>, which in turn will be proceeded on the <receive>. The message will be incorporated into the receiver input of <invoke>, which would make the call to the web service and receive the response to the output jack (approvalInfo). Finally we get the <reply> message which is sent back to the client, and conclude that screenshot of the workflow. Multiple instances of the same process can be run simultaneously. These messages separated with the help of element <correlation>, which is used to route messages to the correct destination.

4.4: EXTENDING THE PREVIOUS EXAMPLE - ADDING LINKS AND DEALING OF DATA

4.4.1: INTRODUCTION

In this section we extend the previous example of the loan approval process, adding links, conditions and assign. Links are used to connect the two actions together and allow one to determine if they will not run another. The conditions are XPath expressions and described how they can handle some conditions about the values of some data. The action <assign> shows how can copy data from one variable to another.

In the previous example, the client sends a loan application which is processed by an organization which sends a response for adding an extra action. If the loan amount is high, then surely the request is sent to the organization to decide. But if the loan amount is low, then a new web service is called, namely 'loan assessor', to determine the risk of the customer. If the loan assessor decides that the risk is low then approves the application, otherwise it sends the request to the organization for overall decision.

4.4.2: CONSTRUCTING THE WORKFLOW

In this BPEL workflow modeled using action <flow>, to decide what action should be parallel and under what conditions will be defined. They allow the <receive> and <invoke> of the previous example and add two <invoke> -one for the assessor and one for the approver. Add an action <assign> to copy the message to reply. After connect the <receive> with two links defined by two conditions: if the amount is less than 10,000 then we call the assessor, otherwise the approver. Then connect the assessor <invoke> with the approver <invoke> with the condition that the risk is high and the <assign> with the condition that the risk is low. Finally, connect the <invoke> of the approver and <assign> with reply, without specifying a condition. The flow that will be followed will be consistent with the conditions that will apply to links.

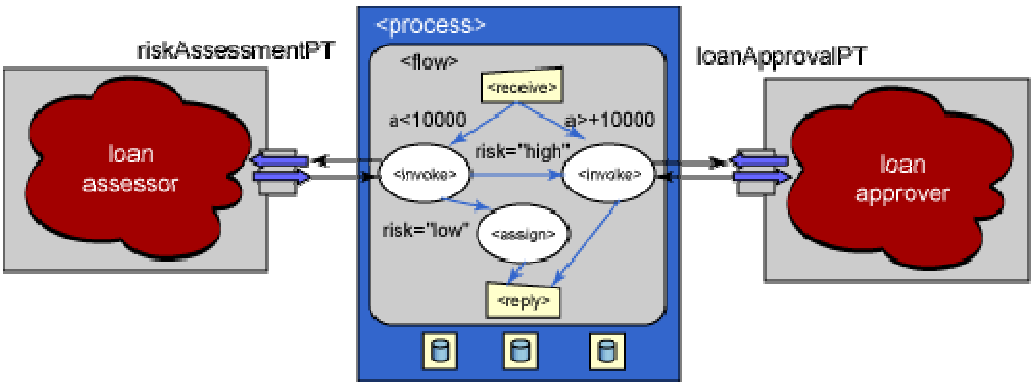


Figure 26: Interior view of the loan approval process

The WSDL description of the loan assessor is shown below. It can only perform an action, in particular 'check', which returns the level of risk.

```
<definitions
   targetNamespace="http://tempuri.org/services/loanassessor"
   xmlns:tns="http://tempuri.org/services/loanassessor"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:loandef="http://tempuri.org/services/loandefinitions"
   xmlns="http://schemas.xmlsoap.org/wsdl/">
<import namespace="http://tempuri.org/services/loandefinitions"
   location=
"http://localhost:8080/bpws4j-
samples/loanapproval/loandefinitions.wsdl"/>
<message name="riskAssessmentMessage">
    <part name="risk" type="xsd:string"/>
  </message>
  <portType name="riskAssessmentPT">
    <operation name="check">
      <input message=
      "loandef:creditInformationMessage"/>
      <output message="tns:riskAssessmentMessage"/>
      <fault name="loanProcessFault" message=
      "loandef:loanRequestErrorMessage"/>
    </operation>
  </portType>
  <binding ...> ... </binding>
  <service name="LoanAssessor">....</service>
</definitions>
```

Figure 27: WSDL description of loan accessor

Add another partnerLinkType in definition of the procedure to determine its interaction with the new partner, the assessor. Each assessor should have the riskAssessmentPT which has been defined earlier.

```
<slnk:serviceLinkType name="riskAssessmentLinkType">
    <slnk:role name="assessor">
      <portType name="asns:riskAssessmentPT"/>
    </slnk:role>
</slnk:serviceLinkType>
```

Figure 28: partnerLinkType

Add one more to partners:

```
<partner name="assessor"
  serviceLinkType="lns:riskAssessmentLinkType"

  partnerRole="assessor"/>
```

Figure 29: assessor partner

In the message receptor add another to get the answer of assessor:

```
<variable
  name="riskAssessment"
  messageType="asns:riskAssessmentMessage"/>
```

Figure 30: Another receptor

Then add the flows and links. The links have a beginning (source) and an end (target). The links have a move condition that takes a Boolean value. If this value be true then followed the link, otherwise the link is not running.

```
      <flow>
          <links>
            <link name="receive-to-assess"/>
            <link name="receive-to-approval"/>
            <link name="approval-to-reply"/>
            <link name="assess-to-setMessage"/>
            <link name="setMessage-to-reply"/>
            <link name="assess-to-approval"/>
          </links>
          <receive name="receive1" partner="customer"
          portType="apns:loanApprovalPT"
                     operation="approve" variable="request"
                     createInstance="yes">
               <source linkName="receive-to-assess"
                     transitionCondition=
                     "bpws:getVariableData('request',
'amount')<10000"/>
              <source linkName="receive-to-approval"
                     transitionCondition=
                     "bpws:getVariableData('request',
'amount')>=10000"/>
          </receive>
         <invoke name="invokeAssessor" partner="assessor"
                 portType="asns:riskAssessmentPT"
```

```
                     operation="check" inputVariable="request"
                     outputVariable="riskAssessment">
            <target linkName="receive-to-assess"/>
            <source linkName="assess-to-setMessage"
                  transitionCondition=
                  "bpws:getVariableData('riskAssessment',
'risk')='low'"/>
            <source linkName="assess-to-approval"
                  transitionCondition=
                  "bpws:getVariableData('riskAssessment',
'risk')!='low'"/>
          </invoke>
```

Figure 31: Flows and Links

If we notice the transition condition bpws: getVariableData ('request', 'amount') <10000 will see that the 'request' is the receiver of a message with type loandef: CreditInformationMessage which contains the variable 'amount'

with type integer. So this treaty appreciates the value of the variable amount if it is less than 10000 then it will return true and will activate the link receive-to-assess. We also have to note condition bpws: getVariableData ('riskAssessment', 'risk') = 'low' which is activated if the risk is low. This needs to be sent back to the client is the answer, so we will use the action<assign>:

```
<assign name="assign">
   <target linkName="assess-to-setMessage"/>
   <source linkName="setMessage-to-reply"/>
   <copy>
        <from expression="'yes'"/>
        <to variable="approvalInfo" part="accept"/>
   </copy>
</assign>
```

Figure 32: Assign action

Finally, add the <invoke> and <reply> like we had before.

```
<invoke name="invokeapprover"
```

```
                partner="approver" portType="apns:loanApprovalPT"
                operation="approve"
                inputVariable="request"
                outputVariable="approvalInfo">
          <target linkName="receive-to-approval"/>
          <target linkName="assess-to-approval"/>
          <source linkName="approval-to-reply" />
        </invoke>
        <reply name="reply" partner="customer"
        portType="apns:loanApprovalPT"
        operation="approve" variable="approvalInfo">
          <target linkName="setMessage-to-reply"/>
          <target linkName="approval-to-reply"/>
        </reply>
      </flow>
</process>
```

Figure 33: Invoke and reply action

It must be clear what happens if a transition condition returns false. This is considered as an error in BPEL and the action which has this condition cast a joinFailure wrong, which if not caught, blocked the entire workflow. To

prevent this we should define a universal property in element suppressJoinFailure <process>. This will stop the extension of joinExceptions and elsewhere. We can see all possible instances of the workflow.

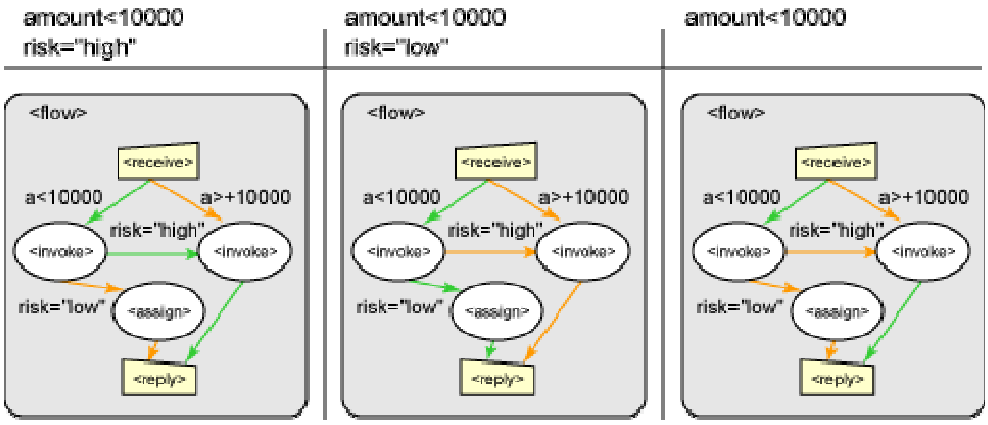


Figure 34: Possible cases of the loan approval process

4.5: EXTENDING THE PREVIOUS EXAMPLE - ADDING CORRELATION AND FAULT HANDLING

We further extend the example of loan approval process, adding the ability to communicate with a previous snapshot of a process and to handle errors that may occur during execution.

4.5.1: INTRODUCTION

If the previous example, approved the loan for a customer then he will probably want to send a second request to obtain the loan. The correlation ensures that this request is addressed to the correct instance of a workflow.

Handling errors will be added to catch an error that states that the client will want to take a loan larger than it was approved.

4.5.2: TAKING THE LOAN: ADDING CORRELATION

To illustrate the correlation, another option should be added to the customer, submit a request to obtain the loan that's approved. We will add an action <receive> to accept the second request of the customer to obtain the loan and a <reply> to disclose that it has acquired. The flow chart shown in the following figure:

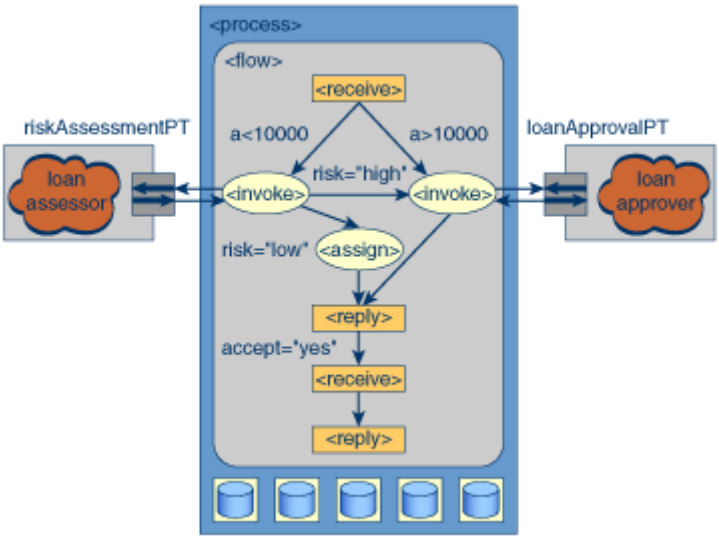


Figure 35: Adding actions to present the correlation

Now we need a way to make sure that the application for obtaining the loan will get the right shot. This is achieved with correlations. To define a relationship we should know the WSDL properties we want to relate. We want them to be the name and surname of the customer, which will give us a unique key for each snapshot. So we define two variables, based on this analysis, the 'applicantFisrtName' and 'applicantLastName'. Then we need to define aliases for these variables so that we can extract the necessary values of the messages being transmitted. We must initialize the correlation set before the second <receive> and we can do the same in each of the actions <invoke>, <receive> action in the first or first <reply> action. However since the approval message has not the correct information, we can exclude the first <reply>. Also since initialize the correlation set is not initialized in both <invoke>, the best point is to initialize the first <receive>. Because both <receive> have the same WSDL message as input, we need to define two <propertyAlias>, one for each message.

```
<bpws:property name="applicantFirstName" type="xsd:string"/>
<bpws:property name="applicantLastName" type="xsd:string"/>
```

```
<bpws:propertyAlias propertyName="lns:applicantFirstName"
messageType=
"loandef:creditInformationMessage" part="firstName" query=
"/firstName"/>

<bpws:propertyAlias propertyName="lns:applicantLastName"
messageType=
"loandef:creditInformationMessage" part-"name" query-"/name"/>

...

<portType name="loanApprovalPT">
<operation name="obtain">
<input message="loandef:creditInformationMessage"/>
<output message="apns:approvalMessage"/>
</operation>
</portType>

<slnk:serviceLinkType name="loanApprovalLinkType">
<slnk:role name="approver">
<portType name="apns:loanApprovalPT"/>
<portType name="lns:loanApprovalPT"/>
</slnk:role>
</slnk:serviceLinkType>
```

Figure 36: Using property alias

Once we define the <property> in the WSDL file, we can define an <correlationSet> in the BPEL file. We will add a new <correlationSet>, called loanIdentifier, at first <receive>. We change the property value for 'initiation' to 'yes', to specify that we want to be initialized at this point. We also add one <correlationSet> to the second <receive> to determine the correct snapshot which will be launched in the message. These are shown in the following figure:

```
<variables>
  <variable                                          name="request"
```

```
messageType="loandef:creditInformationMessage"/>
       <variable name=
       "acceptanceRequest"
messageType="loandef:creditInformationMessage"/>
       <variable                                  name="riskAssessment"
messageType="asns:riskAssessmentMessage"/>
       <variable                                    name="approvalInfo"
messageType="apns:approvalMessage"/>
       <variable                                             name="error"
messageType="loandef:loanRequestErrorMessage"/>
     </variables>

     <correlationSets>
       <correlationSet name="loanIdentifier" properties=
       "lns:applicantFirstName lns:applicantLastName"/>
     </correlationSets>

     ...

     <reply name="initial-reply" partner="customer"
       portType="apns:loanApprovalPT"             operation="approve"
variable="approvalInfo">
       <target linkName="setMessage-to-reply"/>
       <target linkName="approval-to-initial-reply"/>
       <source linkName="reply-to-receive"
         transitionCondition=
         "bpws:getVariableData('approvalInfo', 'accept')='yes'"/>
     </reply>

     <receive name="acceptance-receive"
       partner="customer" portType=
       "lns:loanApprovalPT" operation=
       "obtain" variable="acceptanceRequest">
       <target linkName="reply-to-receive"/>
       <source linkName="receive-to-grant"/>
       <correlations>
         <correlation set="loanIdentifier"/>
       </correlations>
     </receive>
```

```
<reply name="grant-reply" partner="customer"
  portType="lns:loanApprovalPT" operation=
  "obtain" variable="approvalInfo">
  <target linkName="receive-to-grant"/>
</reply>
```

Figure 37: CorrelationSets

These are all the steps needed to add a link in our example. Now, when a message arrives for the second <receive>, the values 'applicantFirstName' and 'applicantLastName' will be compared with the corresponding values stored on all instances of workflows and the message will be routed to the instance that has the same prices.

4.5.3: CORRELATION MESSAGES

Finally the correlation message refers to the location to send a message: to a new or an existing instance of a workflow. Each BPEL correlation is identified by a name and consists of properties that are defined in the WSDL description. The values of properties for each received message determine if an existing snapshot will be delivered or if a new snapshot will be created.

CHAPTER 5

5. WEB SERVICES COMPOSITION

5.1 INTRODUCTION

This chapter aims to review and categorization of web services composition methods, derived from the current standards and bibliography.

5.2 NON-AUTOMATED METHODS AND COMPOSITION LANGUAGES

The description of a web service with WSDL language and other standards for discovery and communication remains at the syntax level, defining messages that provide an abstract description of the data exchanged with the outside world and functions provided by the web service. Based on this description programs - served (client programs) will be created, that communicate with the service by exploiting the interface offered. However, such a simple description is not sufficient for the successful integration of business processes (business processes) based on many web services if it is not supplemented by information on the sequence of operations and the exchange of messages between services. For this reason we must support the ability to form complex business processes that include several web services that work together. The synthesis procedure in this case is referred to as non-automated, because are described as static the structure interactions and the simple services which implement the actions defined in these interactions.

For the description of the necessary interactions that follow well defined business models has emerged a remarkable number of languages and standards which deal with the issue of the description of the composition from a different perspective, but not avoid duplication between them. The most important of these standards are described in the following sections.

5.2.1: BPEL4WS

The language BPEL4WS is a standard model for defining business processes based on Web services using XML. Ousted two earlier models first generation defining workflows based on web services, the XLANG of Microsoft and WSFL of IBM, since the ideas of these two models converge to BPEL4WS. BPEL4WS extends the model of interaction between web services

allowing the description of complex web services corresponding to complex business processes, involving different organizations. The model and the grammar provided by the language define the interactions among multiple services, the functionality offered by the generated composite service and the necessary coordination rules to achieve the realization of business objectives. In addition, it offers the ability to describe situations and logic of interactions, and a systematic way of exception handling (exceptions), i.e. unexpected situations. The definition of BPEL4WS supports both binary (executable) and abstract (abstract) business processes. The executable business processes used to describe the behavior of each of the participants in an interaction. The abstract processes, called business interaction protocols, are used to define public and visible message exchanges between different parties (partners) involved in an interaction, without revealing anything about their internal structure and implementation.

For the execution of complex web services that have been described in BPEL engines the equivalent performance (execution engines) is required. Many companies and organizations have developed such machines, based on a variety of technologies. Among the dominant ones is the IBM WebSphere Business Integration Server Foundation [WBISF], which is a complete BPEL engine that runs on the server platform applications Websphere (WebSphere Application Server) [WAS], and the Oracle BPEL Process Manager [OBPM], which runs on server applications Oracle (Oracle Application Server) [OAS]. There are also open source approaches, such as ActiveBPEL [ActiveBPEL] and bexee [BEXEE]. Finally, there is a particularly light machine Sliver [Hackman et al, 2006], which is designed to run on mobile devices (mobile phones, PDAs), which shows the great interest that exists for the implementation of complex web services and wide spread technology.

Furthermore BPELJ has been suggested, which extends BPEL4WS by allowing Java code within BPEL documents. In enables support functions such calculations values within documents, creation and destruction of documents by using information from other documents, and calculate values for controlling the flow, for example, may form a structure switch which directs the flow depending on the result of a calculation.

The second version of the BPEL language [WS-BPEL2] integrates control structures and branching (if-then-else), and looping constructs (while, repeat-until), so the language natively supports flow control functions. Moreover another important addition is the ability to dynamically call parallel services so as to permit adjustment of the number of steps a process based on the number of participants in this process, giving in this way increased flexibility.

5.2.2: **WSCI**

The WSCI (Web Service Choreography Interface) language is an XML standard for the general description of the interactions between providers and users of web services oriented mostly on message exchanges between the parties involved. This standard defines the flow of messages exchanged between a web service, thus describing the behavior of visible, but does not describe the internal behavior and the functions of the service. For this reason the model is complementary to the static descriptions of functions and defines the standard WSDL, describing how the interaction between them. The description of the interactions of web services is done at two levels. At one level the WSCI describes the messages sent or received by a web service to one process which is a one-sided description concerns only the service itself.

At a second level, however, the description of the externally observable behavior of services is allowed, thereby facilitating the expression of temporal and logical dependencies exchanges messages between different services, so that a service can communicate with one another following the desired cooperation protocols. Finally, it should be noted that the WSCI allows use RDF to add semantic information to the definitions of the interfaces.

The difference of this approach compared to the approach lies in BPEL4WS distributed execution of complex business processes, in contrast to the centralized implementation (centralized execution) in a machine executing BPEL. Participants in choreography interact with each other based on a global (global) scenario, without, as in the case of BPEL, a common point of control for all participants in an interaction.

5.2.3: WS-CDL

The WS-CDL (Web Services Choreography Description Language) language is also based on XML and aims to define the composition of partnerships of long term between providers and users of web services. It provides a general description of the observed global external behavior wherein the order of messages exchanged between all participants in collaboration has resulted in the achievement of a common business goal. Apparently, it can be used to describe abstract processes, regardless of the internal implementation of the same web services. The proposal arise from the observation that in many cases requires the cooperation of many different partners (institutions, independent processes, etc.) in order to achieve a common goal, but the rules of this collaboration is usually described in natural language, while lacking a standardized way of defining the interactions and rules to follow. Using the WS-CDL permit a universal definition of the series of messages and other restrictions on the exchange, and therefore each participant uses this description to develop its services so as to comply with the common rules.

5.2.4: **WSCL**

The WSCL (Web Services Conversation Language) language is another model that allows the definition of the external behavior of services, defining their interactions ("discussions», conversations) at the operational level, and public processes that support. These interactions define the XML documents exchanged as part of the interaction, and the permissible number of these exchanges. The same definitions in WSCL documents are XML therefore can in turn be exchanged between web services. The interaction is usually described from the perspective of the provider of the online service, but this description can be used to extract the corresponding description from the perspective of the service user. The language WSCL achieves the separation of logic from the interaction of business logic and implementation issues of a service, which is why there is the potential to be used in conjunction with complementary description languages such as web services WSDL.

5.2.5: BPML

BPML (Business Process Management Language) [BPML, 2002] is a language for modeling executable and abstract business processes. It is not only applicable for the composition of web services, but generally, integration of business applications and the definition of collaborations with many participants. The BMPL perceive a process as a composition of activities and offers flexibility in defining control flow, data, and events with control structures such as conditional choice (conditional choice), serial (sequential), repetition (iteration) and parallel execution (parallel execution). The BPML completed by the WSDL definition of web services, and has been reported in the literature and the ability to import metadata using semantic standards such as RDF and XHTML, so as to improve the readability of documents by humans and the possibility of understanding and automatic data-processing from the computer.

5.2.6: **BPMN / XPDL**

BPMN (Business Process Modeling Notation) is a graphical representation for the definition of business processes in a workflow. The main element is a Business Process Diagram, and provides the definition of mapping between the graphical notation and the underlying structures in execution languages such as BPEL. BPMN can model both internal business processes, and public abstract processes. Moreover, models collaborative processes (collaboration processes), which involve interactions between two or more partners, which may be implemented through web services.

XPDL (XML Process Definition Language) is an XML standard that has been standardized by the Organization Workflow Management Coalition [WfMC] for the exchange of business process definitions between different tools and systems modeling. Used as a standard exchange diagrams BPMN, because it is designed to completely cover all the elements of language. The XPDL stores information necessary for the execution of a business process, like BPEL, but in contrast to the BPEL stores and graphical information.

5.3: SEMI-automated SYNTHESIS METHODS

In semi-automated approaches attempts are made to automate only a part of the process of web services composition, usually leaving the definition of interaction to the user and decoupling it from the obligation to search among a large number of web services for services that meet the standards and criteria of composition. Identifying appropriate web services to take part in the interaction is not certain in advance, but is a dynamic way. The goal of these approaches is the independence of the implementation from the operational processes. Since the descriptions of business processes are static, the structure of interaction can also be described static and the dynamic part of the process of identifying and selecting individual web services.

Such approaches aim to composition workflows (workflow composition), paralleling a workflow with a complex web service. As the definition of a complex web service includes a set of individual services along with control structures, data flow and messaging between services, similarly the definition of a workflow is to determine the interactions of individual work. So if the problem is treated as synthesis of problem definition, a workflow, current research achievements in the field of automation of workflows can be applied to offer advantages in the process of composition.

In the case of web service composition to support workflow, model abstract processes, which include a set of operations (tasks) and dependencies of their data (data dependencies), must be predetermined by the user before starting the synthesis procedure. The automation is achieved only in the search and selection for each operation of a web service that is able to fulfill their potential while satisfying constraints.

An example system that implements this approach is EFlow. The EFlow is a platform for the definition, standardization and management of complex services in which a composite service is modeled by a graph that defines the order of execution between nodes of the process. The graph may contain service nodes, judgments (decisions) and events. The service nodes represent the calling of an individual or composite service, decision nodes

specify alternative paths and rules governing the execution flow, while event nodes enable services to send or receive different types of events. The graph is not automatically created, but it is defined by the user, so a part of the relatively good knowledge of the scope of the problem. The automation offered by the system for attaching existent services to nodes following the standards set by the user when defining the nodes. To prevent a problem due to non-availability of a personal service, annexation service node needs to be done again each time that one service node, as no advance guarantee the availability of a service at the moment of activation of the corresponding service node.

5.4: AUTOMATED SYNTHESIS METHODS

The approaches described above face the problem of synthesis considering the world static and known, in a business, organization or some general partners. In reality, the world of web services is far from static, since the Internet is open and each provider has the freedom to constantly offer new services or modify existing ones. The management of this large and changing volume web service helps the Semantic Web and the automation of discovery and composition of web services.

Despite efforts to develop common standards describe simple and composite web services, the process of editing and composition has very high complexity due to the following reasons:

- The number of available web services is constantly increasing due to the extremely rapid development of new exposure and easy functionality of existing systems as web services. So the composition process should conduct very extensive search repositories services, which tend to grow rapidly.

- Web services may change several times from the moment of their creation, which in many cases might affect their external interface and then how to contact them. The synthesis process must be able to detect such changes and act accordingly in real time, and many times as a simpler solution proposed to perform the procedure from the beginning.

- Various online services developed by different organizations use different models of concepts (concepts) and natural language to describe them. Existing standards require communication with and support parameter passing syntax checks, but the semantics of these parameters plays a crucial role in the

synthesis. The identification and mapping of these concepts in parameters is a time consuming and difficult process.

It is obvious that the increasing complexity makes the time required for the non-automated synthesis procedure particularly large, so the web service composition should be directed to techniques which favor the automation. These techniques generally fall in the area of Artificial Intelligence and facilitate the automation of the entire process of web services composition, the discovery of appropriate services to individual and production workflow as a final shot composition. However, the implementation of most of these techniques is not possible without semantic information, while the rest are adding semantics facilitates the synthesis process making it more accurate and efficient.

Several models have been developed to express semantic information about web services such as OWL-S [OWL-S] and WSMO (Web Service Modeling Ontology)], which are ontologies describing semantic web services, and SAWSDL (Semantic Annotations for Web Service Description Language) and WSDL-S (Web Service Description Language - Semantics), which is compatible and complementary with the current standard for editorial description of the service WSDL. The model seems to have prevailed among others to add semantic information to web services and is used in many cases these techniques is the OWL-S, a brief description of this model will follow.

The standard OWL-S is an upper ontology based on OWL, which has been created as part of the Semantic Web to describe the knowledge about the semantic web services. The newest version is DAML-S [DAML + OIL]. Used in combination with ontologies, which include the concepts that appear in descriptions of web services in OWL-S, which are either given together with their descriptions, they are part of widespread, general interest ontology such as SUMO (Suggested Upper Merged ontology) or ontologies within the SKOS (Simple Knowledge Organization System). Its use facilitates users, intelligent agents or other software systems to discover, to invite and to compose semantic web services automatically.

More specifically, the development of OWL-S makes the semantics of web services understood by the computer, thus allowing achieving the following objectives:

- Automated discovery of web services: with the development of the Semantic Web more semantic web services are available. The OWL-S will allow the automatic discovery of suitable web service by intelligent agents, based on quality criteria and requirements of the user, without human intervention.

- Automated web service call: currently, call a web service requires the development of a specialized program based on the WSDL description. The OWL-S allows autonomous agents to read the description of a web service, understand the entrances, exits and other requirements and to call it.

- Automated composition of web services: the OWL-S allow interoperability between personal services offered on the network to achieve complex functionality, using only high-level descriptions.

The OWL-S ontology for a web service recognizes three main parts, which are summarized in the follow picture:

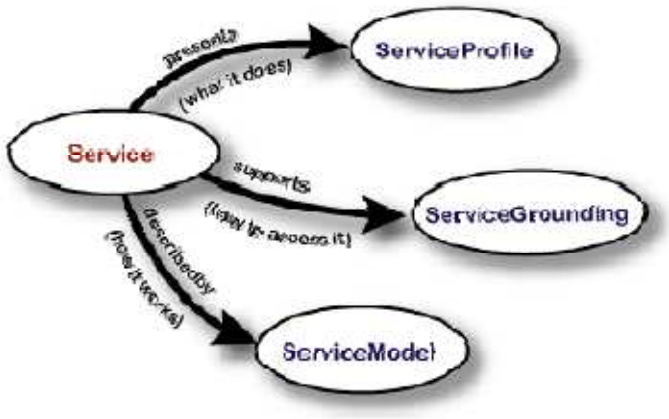


Figure 38: The parts of a web service description in OWL-S

Service Profile: It is used to describe the functionality of a web service and includes information such as name and description, limitations and quality information for the provider.

Service Model: It describes how the client / consumer of the service communicate with it.

Service Grounding: It describes communication details, such protocols and addresses.

Apart from the above mentioned approach for describing semantic web services, OWL-S may be used in addition to the WSDL, if there is already a substantial amount of web service descriptions expressed in this model. The advantage of this approach is the choice of the strengths of each model, i.e. the expressivity of OWL-S in terms of semantics and data types, and support

for WSDL through software systems and transport protocols. The correspondences between the two models are shown in the next figure.

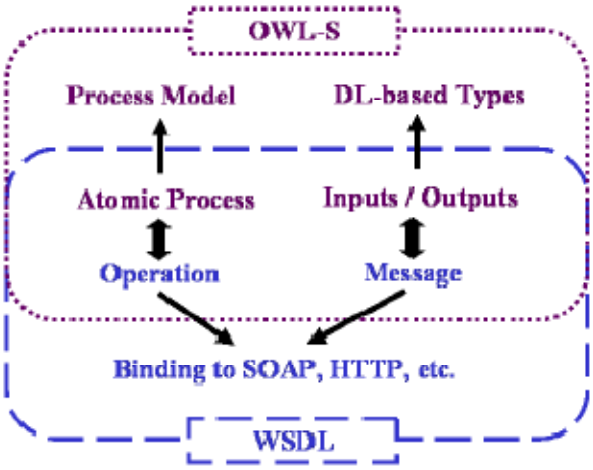


Figure 39: Correspondences between OWL-S and WSDL

Example of a service profile of a web service in OWL-S is shown in the next figure. It contains the name and the description of the service, and information about its parameters. This web service accepts as input parameter the ISBN number of a book, the book is identified and returns as output parameters title and author. The web service belongs to the set of tests of OWLS-TC, which contains the corresponding ontology books in which the concepts belong ISBN, Book Author and used to determine the semantics of the input and output parameters. The description of the service (service profile) provides references to the process model (process model) and information on implementation (service grounding) of the service, as described above, which give additional information on how to call and functioning.

```
<?xml version="1.0" encoding="WINDOWS-1252"?>
<rdf:RDF  xmlns:owl       = "http://www.w3.org/2002/07/owl#"
xmlns:rdfs      = "http://www.w3.org/2000/01/rdf-schema#"
xmlns:rdf       = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:service   = "http://www.daml.org/services/owl-s/1.1/Service.owl#"
xmlns:process   = "http://www.daml.org/services/owl-s/1.1/Process.owl#"
xmlns:profile    = "http://www.daml.org/services/owl-s/1.1/Profile.owl#"
xmlns:grounding = "http://www.daml.org/services/owl-s/1.1/Grounding.owl#"
xml:base        = "http://127.0.0.1/services/1.1/isbn_bookauthor_service.owls">
<owl:Ontology rdf:about="">
<owl:imports rdf:resource="http://127.0.0.1/ontology/Service.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/Process.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/Profile.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/Grounding.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/portal.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/portal.owl" />
</owl:Ontology>
<service:Service rdf:ID="ISBN_BOOKAUTHOR_SERVICE">
<service:presents rdf:resource="#ISBN_BOOKAUTHOR_PROFILE"/>
<service:describedBy rdf:resource="#ISBN_BOOKAUTHOR_PROCESS_MODEL"/>
<service:supports rdf:resource="#ISBN_BOOKAUTHOR_GROUNDING"/>
</service:Service>
<profile:Profile rdf:ID="ISBN_BOOKAUTHOR_PROFILE">
<service:isPresentedBy rdf:resource="#ISBN_BOOKAUTHOR_SERVICE"/>
<profile:serviceName xml:lang="en"> BookProviderService </profile:serviceName>
<profile:textDescription xml:lang="en">
    This service provides a book and its author, on the given ISBN.
</profile:textDescription>
<profile:hasInput  rdf:resource="#_ISBN"/>
<profile:hasOutput rdf:resource="#_BOOK"/>
<profile:hasOutput rdf:resource="#_AUTHOR"/>
<profile:has_process rdf:resource="ISBN_BOOKAUTHOR_PROCESS" /></profile:Profile>
<process:ProcessModel rdf:ID="ISBN_BOOKAUTHOR_PROCESS_MODEL">
<service:describes rdf:resource="#ISBN_BOOKAUTHOR_SERVICE"/>
<process:hasProcess rdf:resource="#ISBN_BOOKAUTHOR_PROCESS"/>
</process:ProcessModel>
<process:AtomicProcess rdf:ID="ISBN_BOOKAUTHOR_PROCESS">
<process:hasInput  rdf:resource="#_ISBN"/>
<process:hasOutput rdf:resource="#_BOOK"/>
<process:hasOutput rdf:resource="#_AUTHOR"/>
</process:AtomicProcess>
<process:Input rdf:ID="_ISBN">
<process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
        http://127.0.0.1/ontology/portal.owl#ISBN</process:parameterType>
<rdfs:label></rdfs:label>
</process:Input>
<process:Output  rdf:ID="_BOOK">
<process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
        http://127.0.0.1/ontology/portal.owl#Book</process:parameterType>
<rdfs:label></rdfs:label>
</process:Output>
<process:Output  rdf:ID="_AUTHOR">
<process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
        http://127.0.0.1/ontology/books.owl#Author</process:parameterType>
<rdfs:label></rdfs:label>
</process:Output>
<grounding:WsdlGrounding rdf:ID="ISBN_BOOKAUTHOR_GROUNDING">
<service:supportedBy rdf:resource="#ISBN_BOOKAUTHOR_SERVICE"/>
</grounding:WsdlGrounding>
</rdf:RDF>
```

Figure 40; An example of service in OWL-S

In the remainder of this section presented techniques have been proposed for automated web service composition, along with indicative systems that have implemented the appropriate technologies. Most systems accept and use the model of OWL-S for semantic description of web services.

5.4.1: DESIGN IN HIERARCHICAL TASK NETWORKS

HTN (Hierarchical Task Network) addresses the problem at various levels of abstraction and are based on a hierarchical decomposition based on networks processes. Entering a system design consists of an initial network processes (task network) that represents the problem to be resolved that the requirements on the composite web service. A network of processes consists of abstract processes defined by other processes. There are three types of processes, primary (primitive tasks) which correspond to simple STRIPS operators and can be executed directly, compound (compound tasks), which are composed of a set of simpler processes, and processes - objectives (goal tasks) are described by conditions as the desired final properties of a situation, in this case a composite web service. Moreover given a set of operators that describe the results of each primary process, i.e. each individual web service, and a set of methods that determine how these complex processes can be decomposed into simpler. The planning process in the first phase decomposes through all the complex processes included in the initial network processes in primary, maintaining constraints, while in the second phase, when the network contains only primary processes, find a plan that satisfies all constraints. This plan is a description of the composite web service consists only of primary procedures, which are mapped to individual web services.

The system SHOP2, which was created as a general purpose system, design activities in hierarchical networks of processes, but later used to compose web services automatically. This differs from other SHOP2 HTN planning systems, because the planning process includes finding the actions that constitute the plan following the series of works as defined in the complex operational process. In this way it becomes possible to know the state of the world at every step of the planning process, and therefore can be applied heuristic techniques or inference techniques to increase efficiency and accelerate the process of composition.

The steps followed in solving a problem of web service composition with SHOP2 is the following:

- **Step 1: Conversion Process Model (Process Models) OWL-S in domains of SHOP2.** In this step, the individual processes are represented by operators SHOP2, and for each complex process created one or more corresponding methods that determine how it decomposes.
- **Step 2: Convert a service composition problem OWL-S in action planning problem SHOP2.** The problem described abstractly as a complex process which must be decomposed into simpler for which there the possibility is mapped in real services.

- **Step 3: Solve the problem and finding a plan by the planning system SHOP2.** The solution to this problem is attempted to be decomposed higher levels of abstract processes simpler, something that typically there are several alternative approaches. At present the information on the ways in which such decomposition is obtained from static descriptions of services in OWL-S.

In addition, the system included subsystem conversion plan produced by the planning system SHOP2 format OWL-S, so there is the possibility of direct execution of the composite service execution and monitoring systems (execution and monitoring systems).

5.4.2: DESIGN ACTIONS ESTIMATE REDUCTION

In these systems the general case the outputs of the desired composite web service is represented as a state - goal the inputs as an initial condition and available individual web services are represented as operators that transform one state to another. All the possible situations that can occur with application an arbitrary number of actions at baseline up the space of states (state space).

The systems designs estimate regression (estimated regression planners) using inverse analysis (backward analysis) the difficulty of the target to guide a proper search (forward search) solution in state space. The search is guided by a heuristic estimator determined using reverse reasoning (backward chaining) to a problem in an area resulting from relaxation, which ignores the interactions between operators that achieve objectives, that completely ignores deletions (deletions). The space created is quite smaller than the state space and a design system may represent the fully through a graph reduction (regression graph).

For their use in the field of web service composition has been proposed to extend traditional symbols such as PDDL. The execution of a service may result in the production of information; however, this is sometimes not properly represented by the structure: effect of PDDL. The proposal made by this approach considers such information as value (value) of an operator, which can act as input to subsequent steps in the plan. For the definition of the extended language PDDL structure including: value in the definition of the operators [McDermott, 2002], which, apart from the actual value of the data returned by a service that has the ability to express these types of data. The new language is created after these changes called Opt, while correspondingly amended the existing designer actions Unpop giving the Optop (Opt-based total-order), which has the potential to handle the new

requirements of the language and used for web services composition problems.

The system implementing the above proposal was used experimentally for simple problems of composition, representing the problem directly in PDDL, while stating the possibility of representation in DAML. However, this approach does not address issues such as scalability and shots conditional or repetitive shots, while research in this area is still in its early stages.

5.4.3: DESIGNED USING MODEL CHECKING

An approach for automated composition of web services by planning on model checking (Planning as Model Checking) attempts to solve the problem using System Planning MBP, after modifications and extensions made to have the ability to handle the case of web services. In the proposed system individual services can be imported and automatically synthesized, as it has been described as abstract processes in BPEL4WS, given a process goal represents the complex service desired by the user, expressed in a suitable language. The final composite service generated by the resolution process is also expressed in BPEL4WS.

The design of model checking system implemented as MBP, and in general, is based on the idea of solving design problems of model actions. The domains are formulated as semantic models (semantic models), their properties as temporal formulas (temporal formulas), while the design is carried out by exploring the state space of the semantic model, and verifying every step of the truth of temporal formulas. The advantages of this approach include the fact that it is well founded, the formulation of the problems is unclear, and are general enough to apply to a wide range of design problems.

This approach seeks to solve design problems by addressing issues such as non-determinism (non-determinism), the partial observability (partial observability) and extended targets (extended goals) , issues in the case of composition of web services often arise. The non-determinism stems from the fact that the planning system effects are not able to predict the actual interactions that take place with external processes, while the partial observability refers to the fact that the planning system does not have full knowledge of the internal state these processes. The objectives are expanded if necessary in many cases the operational requirements define complex conditions that govern the behavior of a whole process and not just the final state.

Despite its advantages, this system is not able to use semantic information during synthesis, which is a serious drawback. Among the disadvantages of the system also include the difficulty scaling to larger and

more complex interaction protocols, because the results on the performance indicate that in such cases the state space may be very large.

Another system that follows this approach is that described in [Yu and Reiff-Marganiec, 2006]. In this work it is recognized the use of technical difficulty with testing design models in web services, where the interaction model is based on messages. Therefore representation of web services with a model based on situations is represented. This is done assuming that each operation of a web service is mapped to a situation which encloses changes after this operation. Apart from the above illustration, the case of the composition of web services it is necessary to take into account other factors such as the flow of messages, business roles and quality requirements. The framework proposed in this approach consists of four phases:

- In the first phase the target (i.e. the desired composite service) is determined, and knowledge about the initial state.
- The second stage is the extraction of the model. The services are the plan automatically selected and is using ontologies to obtain semantic information about them.
- In the third phase the algorithm executed plan is searched for, which can give a set of appropriate plans.
- The fourth and last phases consist of choosing the best plan is translated into executable using the language BPEL, and execution monitoring. In case of failure of performance, selected an alternative plan, if any.

The advantages of this approach include increased scalability and reuse, provided that the plans after their execution are not destroyed but are added to repositories for future use, although web services are changing so quickly that plans were quickly might be considered obsolete. A significant disadvantage with respect to other synthesis approaches through design control models is to use only simple goals.

5.4.4: CALCULUS STATEMENTS

The Golog is a logic programming language based on the situation calculus to define and execute composite operators in dynamic domains. The approach presented in [McIlraith et al, 2001; McIlraith and Son, 2002], attempted to amend the Golog to fit the composition of web services. The problem of the composition is addressed by defining general standards abstract processes that are modified through adaptive constraints. The general idea behind this approach is based on intelligent agents (intelligent agents) who have the ability to perform reasoning over internet services to automatic discovery, execution and composition. User requirements and

constraints are represented with the help of calculus statements. Each web service is assigned to an operator, or primary, which can change the state of the world or to produce any information or composite, which consists of primary effectors. The knowledge base of agents provides a reasonable encoding of the conditions and effects of web services in terms of calculus statements. The agents use a procedural programming language structures to link individual services to create a composite.

The calculus of situations (situation calculus) is a logical formalism designed for representing and reasoning in dynamic domains. It differs from other design techniques actions in the way which is described the state of the world, through the functions f containing relational variables (relational fluents), containing t terms describing a state s: f (t, s). The main components of the software are the operators (actions) that can be applied in the world, the variables (fluents) that describe the state of the world and situations (situations). A domain is described by a set of formulas which include for each operator axioms Conditions for each variable the axioms of sequential state, the axioms that describe the world in various states and the fundamental axioms of the calculus statements.

The descriptions of web services in OWL-S are represented by the calculus of situations, while the composite service is also described in OWL-S. The PDDL used as an intermediate stage between conversions. The Golog used to find a suitable composition up and executed as a program Golog satisfying the properties of the target.

5.4.5: DESIGNED USING RULES

A system that attempts automated design synthesis through actions based on rules (Rule-Based Planning) is the SWORD. The system does not use any of the current standards of web services description, but describes web services indicating the inputs and outputs (preconditions and effects) with the help of the model entity - relationship (Entity - Relationship Model). For each web service created a rule Horn which indicates that the results will be achieved if conditions are true. The synthesis process requires the definition of the initial and final states, and the shot is taken using an expert system based on rules. If the plan created is satisfactory, the user has the possibility to obtain its representation as a composite web service and execution. In cases where the behavior of some services during the execution of the plan deviates from the expected, for example, when a service return multiple alternative outlets, requiring user intervention to resolve the issue before continuing execution.

The system WebDG, which creates complex services automatically using declarative descriptions of high-level, on the theme of e-government, but has been implemented following a general approach to the composition of web services using composition rules (composability rules). This approach is widely used ontologies defined by language DAML + OIL to represent and recognize the semantics of web services. In addition, the system uses composition rules to determine whether two services are composed, relating both syntactic and semantic properties. The syntax rules governing operating modes (operation modes) and the protocols implementation (binding protocols) (subsets of rules called mode composability and binding composability respectively). The semantic rules apply: (1) messages (message composability), comparing data such as the number and type of parameters, ie the compatibility of the costs of a service to the inputs of another, (2) the semantics of operations (operation semantics composability), ie the compatibility between the domains, categories and purpose of the services, (3) quality (qualitative composability), comparing if they match user requirements with quality features offered by web services, and (4) the correctness of composition (composition soundness), which determines whether the composition of some services is reasonable and makes sense of semantic standpoint.

The system works in four phases:

- Definition phase (specification phase), which allows high-level description of the desired complex services using language CSSL (Composite Service Specification Language), which is designated to serve the purposes of this approach.
- Phase Matching (matchmaking phase), in which the composition rules used to create composition plans that conform to the requirements of the user.
- Phase selection (selection phase), in which a shot is selected, if the previous phase are created more than one, based on quality parameters.
- Phase creation (generation phase), which generates a detailed description of the composite service.

5.4.6: HYBRID DESIGN

A system that appeared recently is the OWLS-Xplan, which uses semantic description of web services in OWL-S to create the domain and action planning problem and then tries to obtain a solution calling system design Xplan. This approach is referred to as hybrid design because to solve the problem Xplan system extends the system design FastForward [FF] to a

subsystem planning processes hierarchical networks (HTN) for planning and re-planning when necessary.

To configure the action planning problem, the system uses the semantic description of web services, expressed in OWL-S, along with their corresponding ontologies. However not made full use of the semantic information contained in them, since the use of ontologies not lead to conscious semantic concepts, except to facilitate the formulation of the domain. Then, it can be made semantic relaxation in the design process operations, thus obtaining a solution requires accurate mapping of inputs and outputs of individual web services to the structural level.

The system OWLS-Xplan conforms to some extent to the pattern of PDDL, although the problem is not represented directly with elements of the language. The creators of the system they have developed a dialect of PDDL based on XML, which they named PDDXML, which is used to export the problem of the system and data exchange.

This system seems to be a promising approach in the field of automated composition of web services, mainly due to the future work of the authors, who seem to localize the site of expression of additional, non-functional requirements of the composite web services. However, no data concerning scalability, as well as experiments with the system have been made with a number of domains of effector systems of several tens.

5.4.7: PROOF OF THEOREMS

Another approach to automated web service composition is using automatic theorem proving (Automated Theorem Proving), which is a field of Automated Reasoning (Automated Reasoning) for proving mathematical theorems. In [Waldinger, 2001] available services and user requirements described in a first class language associated with classical logic, and then created the system using evidence proving theorems SNARK [SNARK]. Finally, descriptions of composite web services are exported by specific evidence.

In [Rao et al, 2003] proposed a method for automated composition of semantic web services using theorem proving through linear logic (Linear Logic theorem proving). The method uses the semantic language DAML-S for external representation of web services and internal services are represented by logical axioms and proofs of linear logic. The linear logic allows the formal definition of the properties of web services, and has a close relationship with the p-calculus (p-calculus), which is the standard theoretical basis multilingual composition of web services. In this approach the p-calculus combined with

the rules of linear logic, and model the process of composite web service is generated directly from the proof.

5.4.8: TRANSFORMATION AND SYNTHESIS PROBLEM THROUGH CLASSIC DESIGN ACTIONS

The general idea behind the application of this method in the web service composition is that each web service can be transformed and represented in terms of planning actions directly assigning parts of the OWL-S description of the service elements of a domain and a corresponding problem design effects, or elements PDDL general.

A typical design problem actions generally described by an initial and a final state, and the available operators that allow transitions between states. For each operator must determine the conditions, i.e. the set of events that should apply in the current situation to be able to implement the operator, and its results, which show the state of the world after applying the operator.

In correspondence with the operators, an individual Web service takes some input and produces some output. Therefore, the inputs and outputs of the web service can be represented as preconditions and effects of operators, respectively. An individual line service as an operator, is changing the state of the world after the execution. If the inputs of the desired composite service, that the available data are able to provide the user, as described initial state while the output, namely the effects that would be achieved by the user, as a final state, the design algorithms are actions able to form shots composition automatically without need to have prior knowledge or predefined workflow.

The solution of the direct representation descriptions of web services in OWL-S and PDDL solve the problem of synthesis with different systems design activities compatible with PDDL, regardless of their internal implementation is particularly attractive because of the similarities between OWL-S and PDDL representations. Since OWL-S has been influenced by PDDL, the translation from one language to another is a straight and natural, if used only declarative information. Moreover, with appropriate treatment of ontologies accompanying OWL-S descriptions of web services can be achieved by semantic knowledge representation of the problem. Another big advantage of this approach is that independence to the description of the problem to solve, and requires no modification whatsoever in the systems design actions, unlike many of the aforementioned approaches that extend the current planning system is to ability to deal with the case of web services. This enables the use of any technique or algorithm design activities.

CHAPTER 6

6. WEB SERVICES COMPOSITION WITH HWSC TOOL

6.1 INTRODUCTION

A Web service, w, has typically two sets of parameters: w(i)={I1,I 2,... } for SOAP request (as input) and w(o)={O1,O2,... } for SOAP response (as output). When w is invoked with all input parameters, w(i), it returns the output parameters, w(o). We assume that in order to invoke w, all input parameters in w(i) must be provided. Let's consider a request r with input parameters r(i) and output parameters r(o), the problem to find a set of Web services w∈W such that r(i)⊇w(i) and r(o)⊆ w(o) is referred as Web Service Discovery (WSD) problem. This problem can trivially solve by using a simple look-up table. In this section we focus on the case where no single Web service can fully satisfy the request r and therefore one has to compose multiple Web services. This problem is referred as Web Services Composition problem.

6.2 WEB SERVICES COMPOSITION PROBLEM FORMALIZATION

The problem of interest to the AI community planning relates primarily scenarios that allow interleaving of actions from different sub-projects in a single sequence. By contrast, the WSC problem can be seen as trouble concentrating Information, where Web services are a source of information and entanglement between Web services are not; This allows a very specialized algorithm design. The only requirements for Web services are preconditions knowledge. Furthermore, there are no sibling sub-target interactions. For this reason, a Web service composition algorithm does not model the world state, as do many other designers full AI. Instead of modeling the condition as world of information, a description of information collected by the algorithm at a particular stage in the synthesis.

The Web Services Composition problem can be cast as AI planning problem in STRIPS model. So we can consider a 4-tuple T = < P,W,r(i),r(o) > where :

- P is the parameters set.

- W is the Web Services set
- r(i) ⊆ P are the initial input (request) parameters
- r(o) ⊆ P are the output parameters (response)

A STRIPS model T can be defined as following: T = < S , S(o), S(g), O(.), f, c> where:

- S are collections of parameters in P.
- S(o) ∈ S is such that S(o) = r(i)
- S(g) ∈ S is such that r(o) ⊆ S(g) (goal state).
- O(s) is the collection of the Web Services (w∈W) where w(i) ⊆ s. That means that w can be invoked or applicable in the state s.
- f(w,s)=s' is the function which maps a state s into another state s' where s' = sUw(o) for w∈O(s).
- c(w) is the invocation cost for the Web Service w∈W.

So a finite sequence of Web Services W1,W2,...,Wn such that for a sequence of states S1,S2,...,Sn+1 where Si+1=f(Wi, Si) for i=1,...n, Wi∈O(Si), S(0)∈S1, and S(g)∈Sn+1 consists a solution for the state model.

Based on this solution of STRIPS model, the Web Services Composition Problem can be defined as following: Suppose that a request r has initial input parameters r(i) and desired output parameters r(o). Web Service Composition is to find a finite sequence of Web services,w1,w2,...,wn such that (1) wi can be invoked sequentially from 1 to n, (2) (ri UW(o)1 U... UW(o)n)⊇r(o), and (3) the total cost cost Σ c(Wi) for i=1,…,n is minimized. Consider the .wsdl files in the following figure:

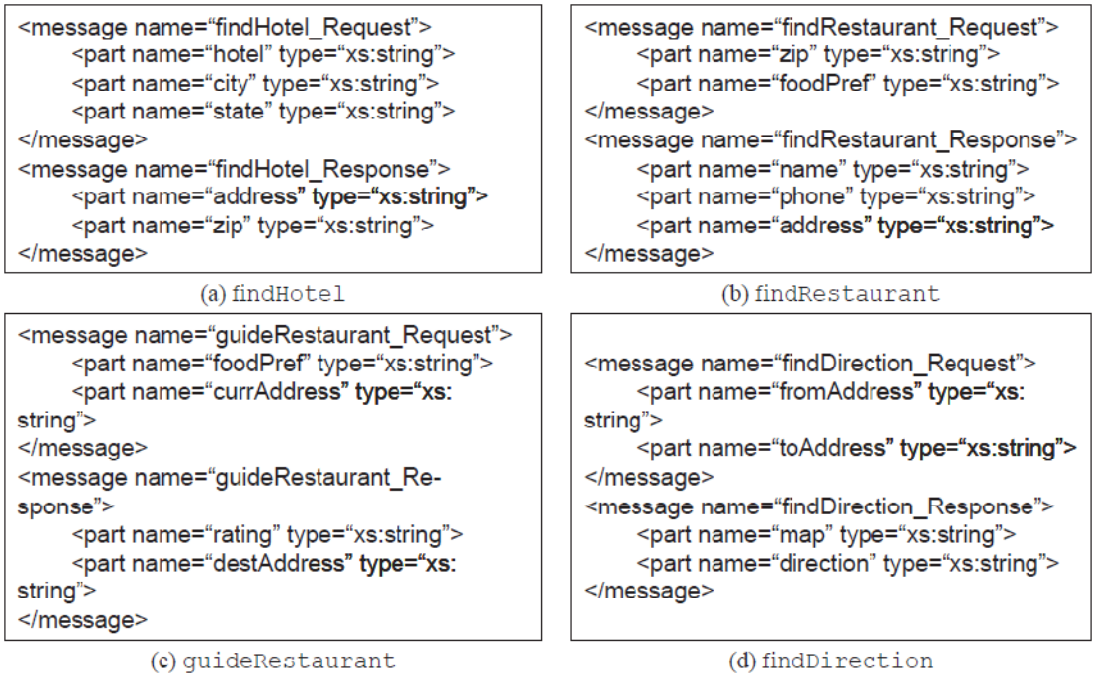
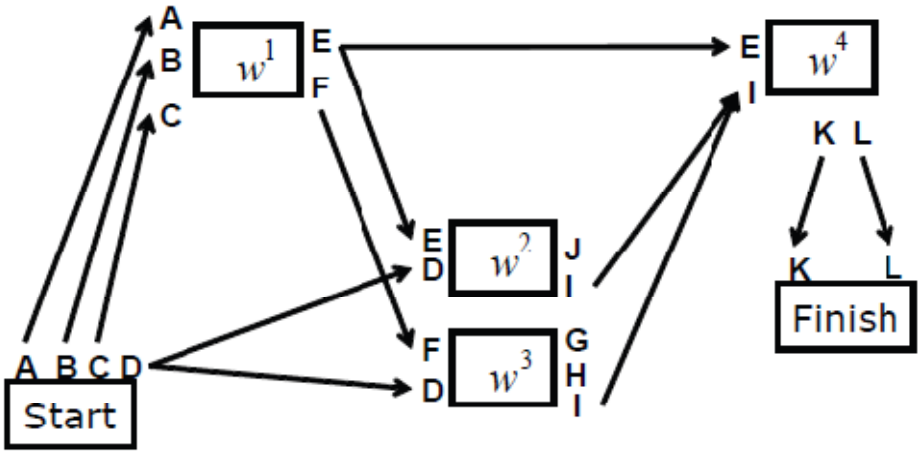


Figure 41: Example WSDL files

Also the r(i) = { “hotelName”,”hotelCity”,”hotelState”,”hotelPreference” } and the r(o)={ “mapHotelRestaurant” , “directionHotelRestaurant“} . The STRIPS model for this example is the following:



P: parameter set

hotelName	A
hotelCity	B
hotelState	C
foodPreference	D
hotelAddress	E
hotelZipcode	F

restaurantName	G
restaurantPhonenumber	H
restuarantAddress	I
restaurantRate	J
mapHotelRestaurant	K
directionHotelRestaurant	L

W: web services set

findHotel	w^1
findRestaurant	w^2
guideRestaurant	w^3
findDirection	w^4

Figure 42: Example STRIPS model

Also the states space for the previous example is in the following figure:

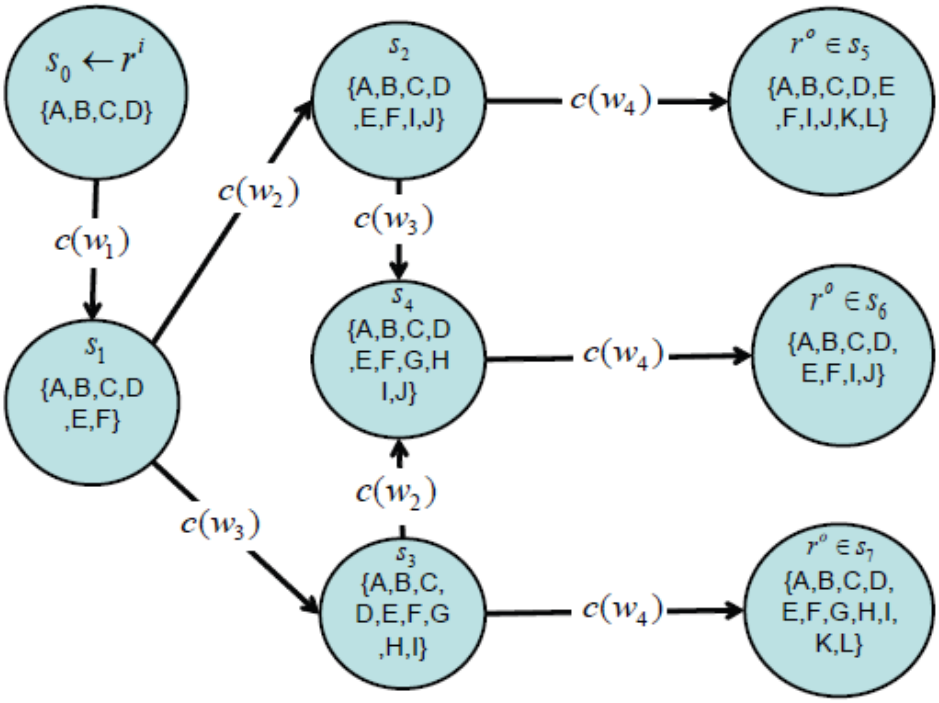


Figure 43: States Space of example

6.3: HWSC ARCHITECTURE

The HWSC tool is divided in three parts. The first part includes the Web Services Data Base which contains 100 Web Services with wsdl description. The second part includes the goal file in specific XML format, in which the user can insert the inputs and the outputs of the request for which wants to export the sequence of the Web Services which compose that. The third part includes the algorithm which export the sequence of the Web Services for a user request, in a BPEL format.

6.3.1: WEB SERVICES DATABASE

In HWSC tool with the term Web Services Data Base we mean the folder which contains the WSDL files of the Web Services. In our implementation we use a folder that contains 100 WSDL files (100 Web Services). The user can give the path for this folder and HWSC read the WSDL files of the folder and load in the memory the inputs and outputs for each Web Service of the specific folder. For saving the inputs and outputs for each Web Service we use an Linked List. In the following figures we can see the previous:

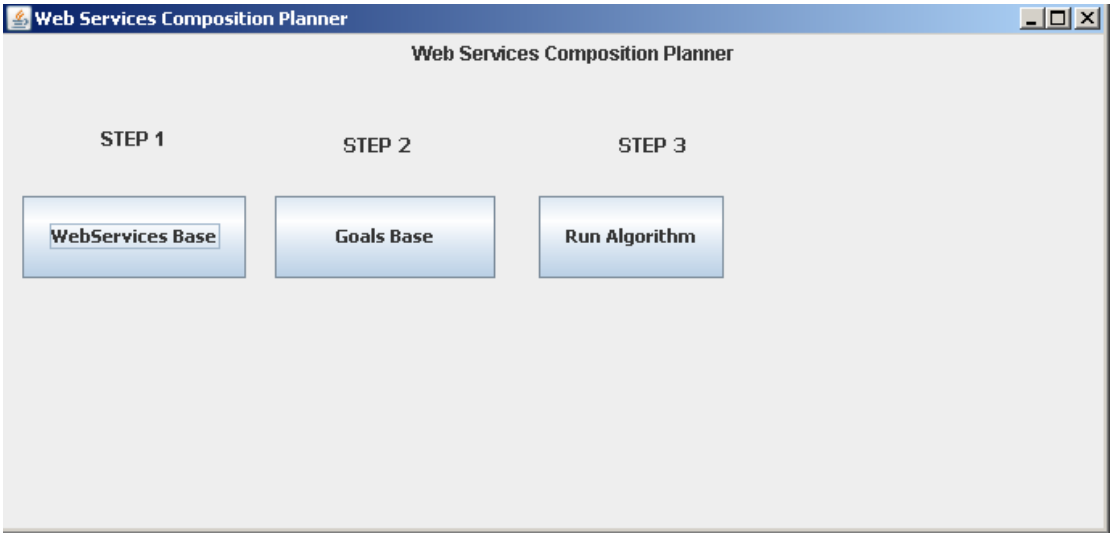


Figure 44: HWSC main window

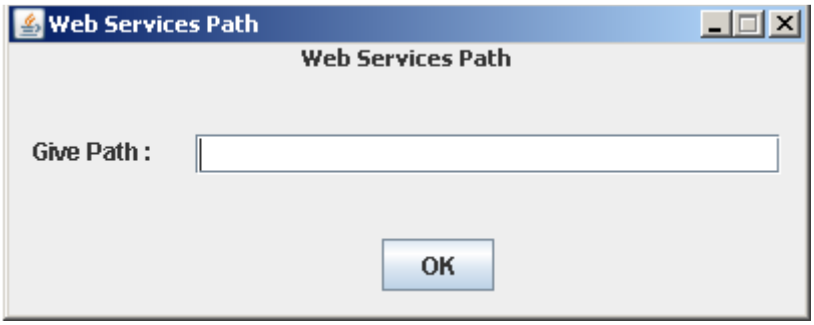


Figure 45: Path for the Web Services Folder

Figure 46: WSDL files in the folder of Web Services

3-->findBanjoInstructor
INPUTS
custFirstName
custLastName
custZipCode
custSkillLevel
banjoStyleToLearn
desiredInstructorGender
OUTPUTS
banjoInstructorName
banjoInstructorID
banjoInstructorEmailAddress
banjoInstructorPhoneNumber
banjoInstructorStreetAddress
banjoInstructorCityAddress
banjoInstructorStateAddress
banjoInstructorZipCode
****
4-->findCloseAPT
INPUTS
custStreetAddress
custCityAddress
custStateAddress
custZipAddress
OUTPUTS
departAirportID
departAirportName
****

Table 1: Load and save inputs and outputs parameters of Web Services

6.3.2: GOALS XML FILE

In the second step the HWSC read the XML file in which the user gives the inputs and outputs parameters. This XML file consists of the following tags:

- **<CompositionRoutine>:** this tag indicates the starting of a composition routine.
- **</CompositionRoutine>:** this tag indicates the end of a composition routine.
- **<Provided>:** this tag indicates that followed belongs in the inputs parameters.
- **</Provided>:** this tag indicates the end of the inputs parameters.
- **<Resultant>:** this tag indicates that followed belongs in the outputs parameters.
- **</Resultant>:** this tag indicates the end of the inputs parameters.

In the following picture we can see goals XML file example:

```
<CompositionRoutine>
        <Provided>"commodityType"</Provided>
        <Resultant>"stockPurchaseConfirmation" </Resultant>
</CompositionRoutine>
<CompositionRoutine>
        <Provided>"custfirstName", "custlastName", "custmiddleInitial", "creditCardNum", "creditCardExp", "creditCardSecID",
        <Resultant> "floristConfirmation", "flowersPurchaseCost" </Resultant>
</CompositionRoutine>
<CompositionRoutine>
        <Provided>  "custFirstName", "custLastName", "custMiddleInitial", "creditCardNum", "creditCardExp", "creditCardSecID"
        <Resultant> "videoStoreConfirmation", "videoRentalCost", "arrivalDateTime", "returnDateTime" </Resultant>
</CompositionRoutine>
<CompositionRoutine>
        <Provided>"forecastCityAddress", "forecastStateAddress","forecastZipAddress","forecastDate"  </Provided>
        <Resultant>"forecastDate","forecastHighTemperature","forecastLowTemperature","forecastCondition", "forecastPrecipitat
</CompositionRoutine>
<CompositionRoutine>
        <Provided> "custStreetAddress","custCityAddress","custStateAddress","custZipAddress","custFirstName","custLastName","
        <Resultant> "mountainConfirmation", "liftTicketPurchaseCost" </Resultant>
</CompositionRoutine>
<CompositionRoutine>
        <Provided> "custStreetAddress","custCityAddress","custStateAddress","custZipAddress","custFirstName", "custLastName",
        <Resultant> "veterinarianConfirmation" </Resultant>
</CompositionRoutine>
```

Figure 47: Goals file example

6.3.3: HWSC ALGORITHM

The main goal of HWSC algorithm is to export a sequence of Web Services which satisfies the user goal. This sequence based on request inputs and outputs that contained in goal XML file. The innovation of HWSC algorithm is the similarity algorithm which uses to read the request inputs and outputs. So the algorithm can run normally in the case of small wrongs in the request inputs and outputs. Specifically gives 20% probability to the user to input wrong request inputs and outputs in the goals XML file. Another

innovation of HWSC is that try to find and the export the smallest sequence of web services that implements the user goal. With this technique we try to reduce the number of web services that are contained in the final sequence. The HWSC algorithm is separated in the following parts:

A) Part 1: Read XML goal file – Similarity Algorithm

In this part the HWSC reads the goals requests from the user goals file and inserts the request inputs in a linked list and the request outputs in another linked list. Firstly, the tool checks if the request inputs and outputs have “wrongs” by using a similarity algorithm. Essentially, checks if the request inputs and outputs there are in the Web Services database, which has been load from the first step of HWSC architecture. Consider the following example to understand the previous:

Suppose that the user gives the request inputs = { firstNameee , lasName , address , city , email } and the request outputs = { occupationnn , salary }. Consider that in the Web Services database there is a Web Service W that has as inputs ={firstName,lastName,address,city,email} and as outputs = { occupation , salary }. The HWSC consider that the user means the same inputs of W even though the user has given wrong inputs and outputs parameters (firstNameee instead of firstName , lasName instead of lastName}.Following the similarity algorithm implementation:

```
public int ComputeDistance(CharSequence str1,CharSequence str2) {

    int[][] distance = new int[str1.length() + 1][str2.length() + 1];

    for (int i = 0; i <= str1.length(); i++)
            distance[i][0] = i;
    for (int j = 0; j <= str2.length(); j++)
            distance[0][j] = j;

    for (int i = 1; i <= str1.length(); i++)
            for (int j = 1; j <= str2.length(); j++)
                    distance[i][j] = minimum(
                                    distance[i - 1][j] + 1,
                                    distance[i][j - 1] + 1,
                                    distance[i - 1][j - 1]
                                                    + ((str1.charAt(i - 1) == str2.charAt(j - 1)) ? 0
                                                                    : 1));

    return distance[str1.length()][str2.length()];
}
```

Figure 48: Compute Distance in similarity algorithm

```
public float GetSimilarity(String string1, String string2){

            float dis=ComputeDistance(string1, string2);
            float maxLen=string1.length();
            if (maxLen < (float) string2.length())
                    maxLen = string2.length();

            float minLen=string1.length();
            if (minLen > (float) string2.length())
                    minLen = string2.length();


            if (maxLen == 0.0F)
                    return 1.0F;
            else
            {
                    //return dis;
                    //return dis/maxLen;
                    //return maxLen - dis;
                    return (1.0F - dis/maxLen)*100 ;
                    //return (float) Math.Round(1.0F - dis/maxLen,1) * 10 ;
            }
    }//telos methodou
```

Figure 49: Similarity Algorithm

The similarity is calculated in 3 steps:

- Partition each string into a list of tokens.
- Computing the similarity between tokens by using a string edit-distance algorithm.
- Computing the similarity between two token lists.

Firstly the algorithm uses an edit-distance string matching algorithm: Levenshtein. The distance of Levenshtein orf SED (String Edit Distance) defined between two strings with random length and is not needed that these strings have the same length. The Levenshtein distance measures the differences between the two strings, not only where there are different letters but even where one string has a character, while the other does not.

For example, the Levenshtein distance between "kitten" and "sitting" is 3, since the following three edits change one into the other, and there is no way to do it with fewer than three edits:

1. **k**itten → **s**itten (substitution of 's' for 'k')
2. sitt**e**n → sitt**i**n (substitution of 'i' for 'e')
3. sittin → sittin**g** (insertion of 'g' at the end).

More specifically, the Levenshtein distance between two strings, s1 and s2, is defined as the smallest number of changes that must occur to the string s1 to be converted to s2. Changes may be made are:

1. Changing a letter in another
2. Adding one letter
3. Deleting a letter

Each of these changes may have a different weight in the calculation of the distance, but in the simplest case of the above three procedures are equal and increase by one unit the distance of a pair of strings. For calculating the distance constructs a table as shown in the following figure:

		-1	0	1	2	3	4	5	6	7	8	9	10
			L	A	W	Y	Q	Q	K	P	G	K	A
-1		0	1	2	3	4	5	6	7	8	9	10	11
0	Y	1	1	2	3	3	4	5	6	7	8	9	10
1	W	2	2	2	2	3	4	5	6	7	8	9	10
2	C	3	3	3	3	3	4	5	6	7	8	9	10
3	Q	4	4	4	4	4	3	4	5	6	7	8	9
4	P	5	5	5	5	5	4	4	5	5	6	7	8
5	G	6	6	6	6	6	5	5	5	6	5	6	7
6	K	7	7	7	7	7	6	6	5	6	6	5	6

Table 2: Levenshtein table

Each of the elements of this table is completed based on the algorithm in figure 46. This table can be calculated line line, as each element of a line depends exclusively from the previous, and in which it is located. Therefore, the complexity of this metric is O (| s1 | * | s2 |), and if that s1, s2 are equal to n, the complexity is O (n^2). It is obvious that the distance of two strings is not dependent on another formation, as well as that there are several alternative routes to convert one string to another with the minimum possible cost. In any case, the Levenshtein distance gives the lowest conversion cost. The reason for which the system uses the algorithm of Levenshtein is that the distance Levenshtein reinforces the similarity between the words which have several common letters even if the letters are not in consecutive positions in the word.

After splitting each string into token lists, we capture the similarity between two strings by computing the similarity of those two token lists, We must notice that the HWSC gives 20% probability to the user to insert wrong inputs and outputs.

Part 2: Main algorithm

The main algorithm of HWSC tool composes Web Services based on the request inputs and outputs. The final composition routine exported in a .bpel template. The algorithm consists of the following parts:

1) **Create Os List (1st part) :** In this part the algorithm creates the O(s) list. O(s) is the list of the Web Services (w∈W) where w(i) ⊆ s. That means that w can be invoked or applicable in the state s. The O(s) list creation based on the transaction function f(w,s)=s' where s'=s U w(o).In the following figure we can see the algorithm for the O(s) list creation . If O(s) size is zero means that cannot find a solution based on the request inputs and outputs of the user.

```
while(checkContains(requestOutputs,tmpS)==0){
        for(i=0;i<ServicesBase.size();i++){
          if(checkContains(((WebServiceElements
)ServicesBase.get(i)).getInputs(),tmpS)==1 && OsList.contains(((WebServiceElements
)ServicesBase.get(i)))==false){
             System.out.println("WebService-->"+((WebServiceElements
)ServicesBase.get(i)).getName());
             OsList.add(((WebServiceElements )ServicesBase.get(i)));
            for(j=0;j<((WebServiceElements
)ServicesBase.get(i)).getOutputs().size();j++){
              if(tmpS.contains(((WebServiceElements
)ServicesBase.get(i)).getOutputs().get(j))==false){
                tmpS.add(((WebServiceElements
)ServicesBase.get(i)).getOutputs().get(j));
              }
            }
            break;
          }

        }
        System.out.println("i-->"+i);
        if(i==ServicesBase.size()){
          return 0;
        }

    }
```

Figure 50: O(s) list creation algorithm

2) **Create final solution list (2nd part) :** In this part the HWSC exports the final composition sequence of Web Services . Firstly in this part the HWSC creates an inverted index (named PD) for all the outputs parameters of Web Services which contained in O(s) list. In the following figure we can see the algorithm which exports the final sequence of Web Services which based on the request inputs and outputs.

```
        sList = (LinkedList)requestOutputs.clone();
        subGoal = (LinkedList)requestOutputs.clone();
        while(checkContains(requestInputs,subGoal)==0){
           System.out.println("requestInputs--->"+requestInputs.toString());
           for(i=0;i<finalIndex.size();i++){
              tmp = ((myIndexElements)finalIndex.getMyIndex().get(i)).getPName();
              if(subGoal.contains(tmp)==true){

wSpace.add(((WebServiceElements)((myIndexElements)finalIndex.getMyIndex().get(i)).
getWs()));
              }

           }
           x = getMaxWs();
           System.out.println("MAX-->"+x.getName());
           for(j=0;j<x.getOutputs().size();j++){
              if(sList.contains(x.getOutputs().get(j))==false){
                 sList.add(x.getOutputs().get(j));
              }
           }
           for(j=0;j<sList.size();j++){
             if(requestInputs.contains(sList.get(j))==true){
                 sList.remove(j);
                 j=0;
              }
           }
           for(j=0;j<x.getInputs().size();j++){
              if(subGoal.contains(x.getInputs().get(j))==false){
                 subGoal.add(x.getInputs().get(j));
              }
           }
           for(j=0;j<subGoal.size();j++){
              if(sList.contains(subGoal.get(j))==true){
                 subGoal.remove(j);
                 j=0;
              }
           }
           if(soln.contains(x)==false){
              soln.add(x);
              help.add(x.getName());
           }
           System.out.println("soln--->"+help.toString());

        }
```

Figure 51: Algorithm that exports the final composition sequence of Web Services

The algorithm based on a forward search. The forward search mentioned in the bibliography as regression search and is an old idea in ai planning. In regression search, a state s can be thought as a set of effects and we can specify a sub-goal from the state s. In the algorithm implementation sub-goal setting to be r(o) \ r(i) at the starting . Also, wSpace is setting to be a list of Web Services such that $w(i) \in PD_{ws}(p)$, $p \in subGoal$. So, the algorithm selects a Web Service from wSpace by considering their heuristics at each backward step. The repeated procedure of algorithm are executed until $subGoal \subseteq r(i)$.

The innovation of algorithm is the reduce number of Web Services which are contained in the final composition sequence. For this innovation we based on the following hypothesis:

- If we choose a Web Service with bigger contribution to match the subgoal earlier in the regression search, it helps to reach to the initial state faster.

To implement the previous hypothesis in our algorithm we set a parameter h(w)= | w(o) ∩ subGoal | which helps the algorithm to find the Web Service which has the max h(w). If a Web Service has the max h(w) can have the chance to preventing proliferation of search space. For exporting the final solution the tool use the following repeated procedure:

```
tmpSlist = (LinkedList) requestInputs.clone();
tmpSoln = (LinkedList) soln.clone();
for(i=0;i<ServicesBase.size();i++){
  if(checkContains(((WebServiceElements)ServicesBase.get(i)).getInputs(),tmpSlist)==1 && tmpSoln.contains((WebServiceElements)
ServicesBase.get(i))==true){
    if(finalSolution.contains(((WebServiceElements)ServicesBase.get(i)))==false){
      finalSolution.add(((WebServiceElements)ServicesBase.get(i)));
      for(j=0;j<((WebServiceElements)ServicesBase.get(i)).getOutputs().size();j++){
        if(tmpSlist.contains(((WebServiceElements)ServicesBase.get(i)).getOutputs().get(j))==false){
          tmpSlist.add(((WebServiceElements)ServicesBase.get(i)).getOutputs().get(j));
        }
      }
      i=-1;
    }

  }
  if(finalSolution.size()==tmpSoln.size()){
    break;
  }
}
System.out.println("finalSol size-->"+finalSolution.size());
System.out.print("finalSolution-->");
for(i=0;i<finalSolution.size();i++){
  System.out.print(((WebServiceElements)finalSolution.get(i)).getName()+",");
}
```

Figure 52: Repeated procedure for final solution

The procedure selects the sequence of Web Services by following the next algorithm:

1) Firstly check if each w that are contained in soln list are contained in the O(s) list.
2) If contained set s = s U w(o)
3) Else stop procedure
4) When the variable which checks if the soln has been read, be zero, the procedure stop and print the final solution.

6.3.4: EXAMPLE

In this section we present some examples of using HWSC tool to create a composition sequence of Web Services based on request inputs and outputs for each example.Consider the following request inputs and outputs:

r(i) = { “hotelName”,”hotelCity”,”hotelState”,”hotelPreference” }

r(o)={ “mapHotelRestaurant” , “directionHotelRestaurant“}

Also consider the following Web Services:

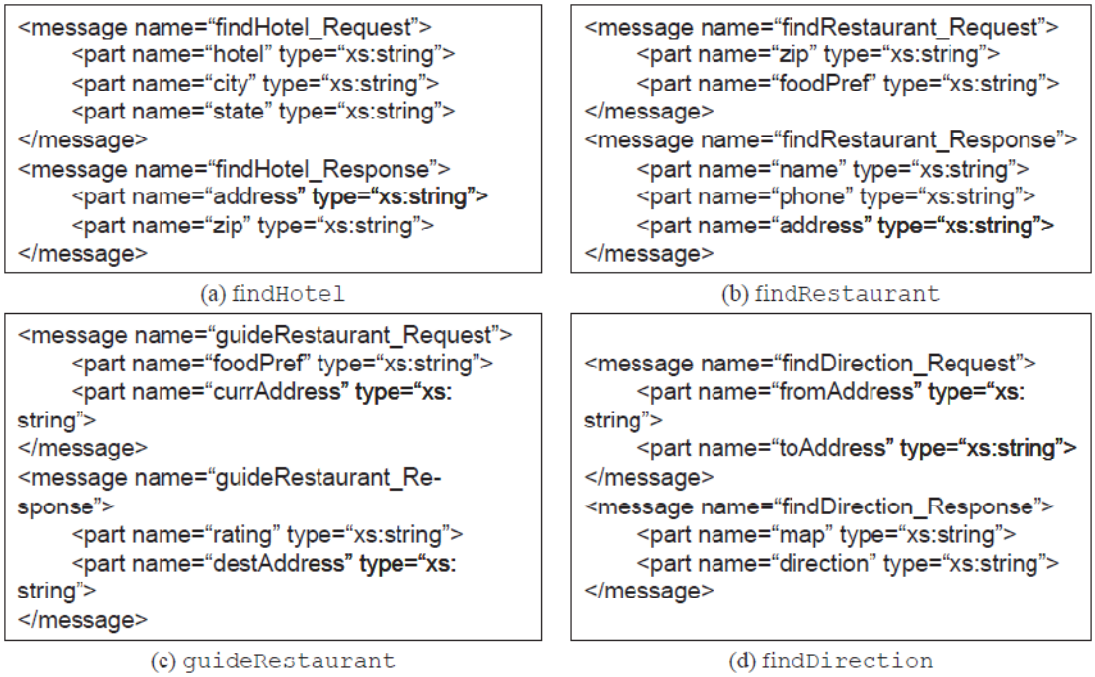


Figure 53: WSDL files of Web Services Example

The first part of composition algorithm consists of the following steps:

1) s = { “hotelName”,”hotelCity”,”hotelState”,”hotelPreference” }
   O(s) = { findHotel }
   s = s U {“hotelAddress”,”hotelZipCode”}
2) O(s) = { findHotel” ,”findRestaurant” }
   s=s U {“restaurantRate”,”restaurantAddress”}
3) O(s) = { “findHotel” ,”findRestaurant” ,”findDirection”}
   s = s U {“mapHotelRestaurant”,”directionHotelRestaurant”}
4) Finish because s ⊆ r(i)

The second part of composition algorithm consists of the following steps:

1) PD(hotelAddress) = {findHotel}
   PD(hotelZipcode) = {findHotel}
   PD(restaurantAddress) = {findRestaurant}
   PD(restaurantRate) = {findRestaurant}
   PD(mapHotelRestaurant) = {findDirection}
   PD(directionHotelRestaurant) = {findDirection}

2) s = { “mapHotelRestaurant” , “directionHotelRestaurant“}
   subGoal = { “mapHotelRestaurant” , “directionHotelRestaurant“}
   PD(mapHotelRestaurant) = {findDirection}
   PD(directionHotelRestaurant) = {findDirection}
   X = {findDirection}

s = s U {“ restaurantAddress”,” hotelAddress”}
subGoal = {“ restaurantAddress”,” hotelAddress”}
soln = {findDirection}

3) PD(restaurantAddress) = {findRestaurant}
PD(hotelAddress) = {findHotel}
wSpace = {“ findHotel” , “findRestaurant”}
Because each web service has h = 1 we randomly can select
X = {“findHotel”}
s = s U {“hotelAddress”,”hotelZipCode”}
subGoal = { “ restaurantAddress”}
soln = {findHotel,findDirection}

4) PD(restaurantAddress) = {findRestaurant}
wSpace = {“findRestaurant”}
s = s U { “restaurantName”, restaurantPhone”, “restaurantAddress”}
subGoal = { }
soln = {findHotel,findDirection,findRestaurant}

5) s’ = { “hotelName”,”hotelCity”,”hotelState”,”hotelPreference” }
finalSolution = { findHotel } because the web service which satisfies the s’ is the findHotel.
s’ = s’ U {“hotelAddress”,”hotelZipcode”}
finalSolution = { findHotel , findRestaurant} because the web service which satisfies the s’ is findRestaurant
s’ = s’ U {“restaurantAddress”, “ restaurantRate“}
finalSolution = { findHotel , findRestaurant , findDirection} because the web service which satisfies the s’ is findDirection.
Procedure stops because the soln list has been read and the finalSolution contains the final composition sequence.

CHAPTER 7

7. EXPERIMENTS

In this chapter we present some results of the test which executed with the HWSC tool. The computer which used in the experiments was a dual core T2390 1.86GHz with 3GB RAM and Windows 7 Professional 32-bit as operating system.

7.1: LOAD WEB SERVICES IN MEMORY

In the following figures we can see the time which required loading in the memory the Web Services with your parameters.

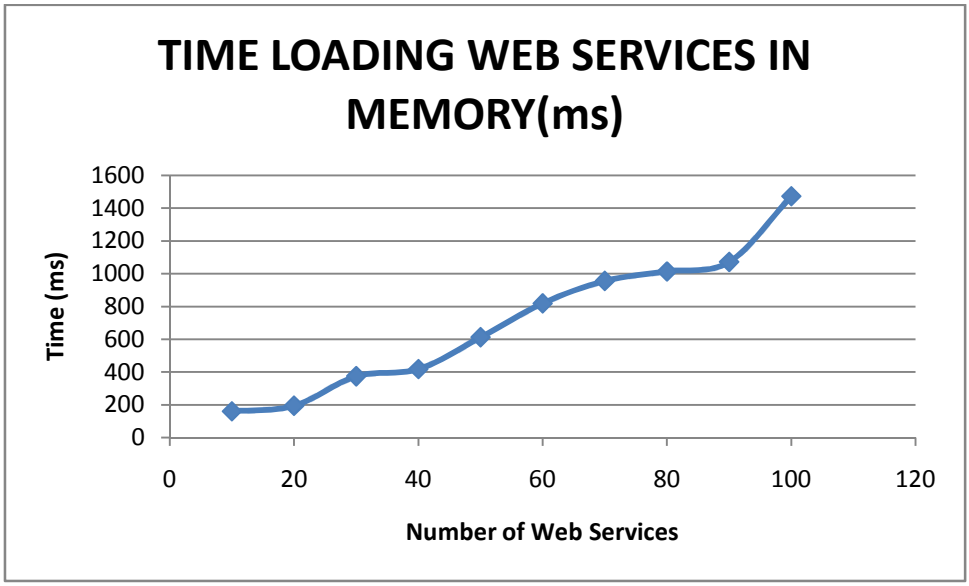


Figure 54: Time loading web services in memory

#W	TIME LOAD IN MEMORY(ms)
10	160
20	194
30	374
40	418
50	612
60	819
70	956
80	1014
90	1072
100	1473

Table 3: Comparison of loading 10 sets of Web Services in memory

From the previous we can observe that there is a linear increase in loading time of Web Services in memory. Also we can easily see that when the number of Web Services increases, the time loading of services in memory, also increases.

7.2: O(s) TIME CREATION

In the following table we can see the results of creation O(s) list for different goals.

No.GOAL	#REQUEST INPUTS	#REQUEST OUTPUTS	#w in sequence	O(s) Creation time(s)
1	12	1	4	0,001121036
2	1	1	4	0,000627188
3	1	1	3	0,001494532
4	10	2	1	0,000246193
5	12	4	2	0,000243998
6	4	6	1	0,000215647
7	11	2	2	0,004594987
8	8	1	2	0,008825623
9	9	2	1	0,003841777
10	3	1	3	0,000428734

Table 4: O(s) creation time

From the previous table we can conclude that there is a correlation between the time of finding the final sequence of web services and the number of request inputs and outputs. Also, we can see that the longer times there are in cases where the final composition sequence consists of more than one web services. In goal No.9 we have long time because many of request inputs of this goal there are in many web services, so required more time for the O(s) creation.

7.3: TESTING DIFFERENT GOALS

In the following table we can see the results of execution time for different goals. We run 10 goals where each goal had at least one request input and output.

No.GOAL	#REQUEST INPUTS	#REQUEST OUTPUTS	#w in sequence	TIME(ms)
1	12	1	4	73
2	1	1	4	45
3	1	1	3	29
4	10	2	1	5
5	12	4	2	316
6	4	6	1	3
7	11	2	2	277
8	8	1	2	354
9	9	2	1	249
10	3	1	3	8

Table 5: Testing results for 10 goals

From the previous table we can conclude that there is a correlation between the time of finding the final sequence of web services and the number of request inputs and outputs. Also observe that in the case where the final sequence consists of one web service the final time is smaller than in the case where the final sequence consists of more than one web services. So, the number of web services that are contained in final sequence, affect the final composition time.

In the following table we can see the results of composition time in the case that goals XML file contains more than one goal. Easily someone can see that there is a linear increase in the final composition time.

#GOALS	TIME(ms)
1	194
2	317
3	431
4	490
5	678
6	917
7	1441
8	1663
9	2065
10	2301

Table 6: Comparison goals composition time

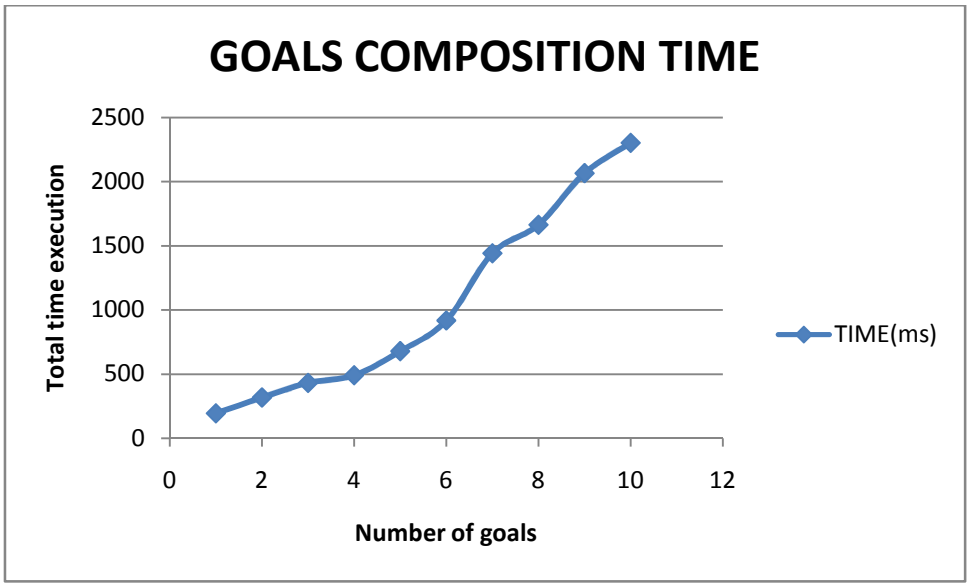


Figure 55: Goals composition time

CHAPTER 8

8. CONCLUSIONS AND FUTURE WORK

8.1: CONCLUSIONS

A Web service is typically characterized by two sets of parameters: a set of inputs (usually represented by a SOAP request) and a set of outputs (usually represented by a SOAP response). A successful execution of a Web service with all required input parameters (which are assumed to be available) is expected to produce the required output parameters. Given a request containing a set of input parameters and requesting a set of output parameters, a typical problem is to find candidate single Web services that can produce the required output if the required input is provided. This problem is defined as Web Service discovery in literature.

In this thesis we focus on the case where there is not any single Web service that fully satisfies a given request. In this case we must compose multiple Web services in order to satisfy the given request. Web Service composition aims to address that problem. Web Service composition involves combining and coordinating a set of services in a workflow with the purpose of achieving functionality that cannot be realized with any individual service. Several service composition models have been proposed, with the most prominent ones being service orchestration and service choreography.

The web service composition is particularly useful for business process integration, and where the functionality offered by individual services is not enough. However the synthesis procedure is expected to become more and more difficult, time consuming and inefficient, as the complexity of the problem formulation is increasing with the increasing number of available individual web services. For this reason, the ability to automate the process of composition proved an important factor for the survival of web services in business.

We presented a tool named HWSC which reads a request expressed by a use in a predefined format and attempts to answer the request by creating a composite service combining Web services from a repository. The tool first reads the WSDL descriptions of the Web Services contained in the

repository and then applies a composition algorithm that results in a composite service that satisfies the request. The innovative characteristics of the tool are: 1) Provides a lightweight approach to composition based only on WSDL input-output descriptions. 2) Uses similarity techniques while reading a user request allowing for a 20% probability of error in input and output names. 3) Uses an algorithm based on STRIPS, an automated planner in artificial intelligence, in order to find the composition sequence that satisfies the user request. 4) Exports the final composition sequence in a BPEL template file, which can be edited by tools supporting BPEL and executed by a BPEL engine.

8.2: FUTURE WORK

One of the future work of this thesis is to use indexes for faster storage of parameters of web services. This will have as result the reduction of access time of the repository of web services, resulting in faster and more responsive to the export of which involves the faster export of the final composition sequence.

Another future work consists the using of neural networks. By using the model of neural network we can train different neurons for specific composition plans so it will export the final composition sequence of web services. The neural network model can give the possibility to make the tool to be more functional because the neurons (if they train) can simulate the human thinking.

Also, another area in which can focus the future work is the improvement the BPEL template of the final composition sequence by adding JAVA code in the BPEL code. With JAVA code we can implement each web service so the final BPEL will be executed immediately without additional processing.

The tool can be more functional offering the ability to support more complex wsdl files and by supporting OWL-S files which describe web services by using OWL syntax.

Finally an additional future work is the visualization of the final composition sequence of web services which are contained in final bpel file. The visualization of final result helps understanding the problem and improves the presentation of final result.

CHAPTER 9

9. BIBLIOGRAPHY

[1] V. Alevizou and D. Plexousakis. Enhanced specifications for web service composition.In ECOWS, pages 223{232. IEEE Computer Society, 2006.

[2] Altintas, I., Jaeger, E., Lin, K., Lu daescher, B. & Menon, A. (2004)).A Web Service Composition and Deployment Framework for Scientific Workflows. Proceedings of the 2nd IEEEEEEEEE International Conference on Web Services (ICWS), San Diego, CA.

[3] Blum, A. & Furst, M.(1997)Fast planning through planning graph analysis. Artificial Intelligence,90, 281- 300.

[4] Bylander,T.(1994).The computational complexity of propositional STRIPS planning, Artificial Intelligence, 69(1- 2),

[5] Carman, M., Serafini L. & Traverso, P.(2003).Web service composition as planning. Proceedings of the 13th International Conference on Automated Planning.

[6] Constantinescu, I., Faltings, B. & Binder, W. (2004)).Large Scale Testbed for Type Compatible Service Composition. Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAP S), Whistler, British Columbia, Canada.

[7] Dong, X., Halevy, A., Madhavan, J., Nemes, E. & Zhang, J. (2004). Similarity Search for Web Services. Proceedings of the 30th Very Large Data Bases (VLDB)VLDB)VLDB)VLDB)VLDB), Toronto, Ontario, Canada.

[8] Ghandeharizadeh, S. et al.,)(2003).Proteus: A Sy stem for Dynamically Composing and Intelligently Executing Web Services. Proceeding of 1the 1st IEEE International Conference on Web Services (ICWS), Las Vegas, NV.

[9] Oh, S.-C., Lee, D. and Kumara, S. (2005), A comparative illustration of AI planning-based Web services composition, ACM SI Gecom Exchanges, 5(5), 1-1 0.

[10] Oh, S.-C., On, B., Larson, E. J. & Lee, D. (2005) BF*: Web Services Discovery and Composition as Graph Search Problem. Proceedings of the 7th International IEEE -Conference on e-Technology, e-Commerce and e-Service (EEE) , Hong Kong, China.

[11] Ponnekanti, S. R. & Fox, A. (2002). SWORD: A developer toolkit for Web service composition. Proceedings of the 1111th World Wide Web (WWS),Honolulu, HI.

[12] Rao, J. & Su, X.(2004).A Survey of Automated Web Service Composition Methods. Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC) , San Diego, CA.

[13] T. Andrews et al. Business Process Execution Language for Web Services (BPEL4WS) 1.1. Online: http://www.106.ibm.com/ developerworks/ webservices/ library/ws-bpel, May 2003.

[14] T. Bellwood et al. Universal Description, Discovery and Integration specification (UDDI) 3.0. Online: http://uddi.org/pubs/uddi-v3.00-published-20020719.htm.

[15] D. Box et al. Simple Object Access Protocol (SOAP) 1.1. Online: http://www.w3.org/TR/SOAP/, 2001.

[16] R. Chinnici et al. Web Services Description Language (WSDL) 1.2. Online:http://www.w3.org/TR/wsdl/.

[17] S. R. Ponnekanti and A. Fox. SWORD: A developer toolkit for Web service composition. In Proceedings of the 11th World Wide Web Conference, Honolulu, HI, USA, 2002

[18] J. Rao, P. K¨ungas, and M. Matskin. Logic-based Web services composition: from service description to process model. In Proceedings of the 2004 International Conference on Web Services, San Diego, USA, July 2004. IEEE.

[19] H. Schuster, D. Georgakopoulos, A. Cichocki, and D. Baker. Modeling and composing service-based and reference process-based multi-enterprise processes. In Proceeding of 12th International Conference on Advanced Information Systems Engineering (CAiSE), Stockholm, Sweden, June 2000. Springer Verlag.

[20] D. Wu, E. Sirin, J. Hendler, D. Nau, and B. Parsia. Automatic Web services composition using SHOP2. In Workshop on Planning for Web Services, Trento, Italy, June 2003.

[21] Agarwal, V.; Chafle, G.; Dasgupta, K.; Karnik, N.; Kumar, A.; Mittal, S.; and Srivastava, B. 2005. Synthy: A system for end to end composition of web services. JWS 3(4).

[22] Bryce, D.; Kambhampati, S.; and Smith, D. E. 2006. Planning graph heuristics for belief space search. JAIR 26:35–99.

[23] Eiter, T.; Faber,W.; Leone, N.; Pfeifer, G.; and Polleres, A. 2003. A logic programming approach to knowledge-state planning, II: The DLVK system. AI 144(1-2):157–211.

[24] Hoffmann, J., and Brafman, R. 2006. Conformant planning via heuristic forward search: A new approach. AI 170(6–7):507–541.

[25] Fensel, D.; Lausen, H.; Polleres, A.; de Bruijn, J.; Stollberg, M.; Roman, D.; and Domingue, J. 2006. Enabling Semantic Web Services– The Web Service Modeling Ontology. Springer-Verlag.

[26] Hoffmann, J., and Brafman, R. 2006. Conformant planning viaheuristic forward search: A new approach. AI 170(6–7):507–541.

[27] Hoffmann, J.; Bertoli, P.; and Pistore, M. 2007. Web service composition as planning, revisited: In between background theories and initial state uncertainty. Technical report, DERI Innsbruck. Available at http://members.deri.at/ joergh/papers/tr-aaai07.ps.gz.

[28] Pistore, M.; Traverso, P.; and Bertoli, P. 2005. Automated composition of web services by planning in asynchronous domains. In Proc. ICAPS-05.

[29] Sirin, E., and Parsia, B. 2004. Planning for semantic web services. In Workshop “Semantic Web Services” at ISWC-04.

[30] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju,Web Services: Concepts, Architectures and Applictions, 2004.

[31] Biplav Srivastava and Jana Koehler, ‘Web service composition – current solutions and open problems’, in ICAPS 2003 Workshop on Planning for Web Services, (July 22 2003).

[32] Blake, M.B. An Agent-Based Cross-Organizational Workflow Architecture in Support of Web Services, WETICE 2002: 176-181

[33] Casati, F., Ilnicki, S. and Jin, L Adaptive and Dynamic Service Composition in eFlow. HP Technical Report, HPL-2000-39, March, 2000, www.hpl.hp.com/techreports/2000/HPL-2000-39.pdf

[34] Yang, J and Papazoglou, M. Web Component: A Substrate for Web Service Reuse and Composition. in Procs of the 14th International Conference on Advanced Information Systems Engineering (CAiSE02), May, Toronto, Lecture Notes in Computer Science, Vol. 2348, p21-36, Springer, 2002

[35] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klien, Frank Leyman, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic and Snajiva Weerawarana, "Business Process Execution Language for Web Services"

[36] Fahiem Bacchus and Froduald Kabanza, "Using Temporal Logics to Express Search Control Knowledge for Planning," *Artificial Intelligence*, vol. 116, no. pp. 123-191, 2000.

[37] Prashant Doshi, Richard Goodwin, Rama Akkiraju and Kunal Verma, "Dynamic Workflow Composition: Using Markov Decision Processes," *International Journal of Web Service Research*, vol. 2, no. 1, pp. 1-17, 2005.

[38] Kutluhan Erol, James Hendler, and Dana S. Nau, "HTN Planning: Complexity and Expressivity," presented at Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), 1994.

[39] Malik Ghallab, Dana Nau, and Paolo Traverso, *Automated Planning: Theory and Practice*, Amsterdam Morgan Kaufmann Publisher, 2004.

[40] Fausto Giunchiglia and Paolo Traverso, "Planning as Model Checking," presented at Proceedings of the 5th European Conference on Planning: Recent Advances in AI Planning, London, UK, 1999.

[41] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra, "Planning and Acting in Partially Observable Stochastic Domains," *Artificial Intelligence*, vol. 10, no. 1- 2, pp. 99-134, 1998.

[42] Ugur Kuter, Dana Nau, Marco Pistore and Paolo Traverso, "A Hierarchical Task-Network Planner based on Symbolic Model Checking", in *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling*, 2005, pp. 300-309.

[43] Ugur Kuter and Dana S. Nau, "Forward-Chaining Planning in Nondeterministic Domains", in *Proceedings of the AAAI*, 2004, pp. 513-518.

[44] Hector J. Levesque, Raymond Reiter, Yves Lesperance, Frangzhen Lin and Richard B. Scherl, "GOLOG: A Logic Programming Language for Dynamic Domains," *Journal of Logic Programming*, vol. 31, no. 1-3, pp. 59-83, 1997.

[45] [J. Magee and J. Kramer, *Concurrency - State Models and Java Programs*, John Wiley,1999.

[46] David Martin, Anupriya Ankolekar, Mark Burstein, Grit Denker, Daniel Elenius, Jerry Hobbs, Lalana Kagal, Ora Lassila, Drew McDermott, Deborah McGuinness, Sheila McIlraith, Massimo Paolucci, Bijan Parsia, Terry Payne, Marta Sabou, Craig Schlenoff, Evren Sirin, Monika Solanki, Naveen Srinivasan, Randy Washington. "OWL-S 1.1 Release," 2004.

[47] Drew V. McDermott, "Estimated-Regression Planning for Interactions with Web Services", in *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems*, 2002, pp. 204-211.

[48] Sheila McIlraith and Ronald Fadel, "Planning with Complex Actions", in Proceedings of the 9th International Workshop on Non-Monotonic Reasoning, 2002, pp. 356-364.

[49] Dana Nau, Yue Cao, Ammon Lotem and Hector Munoz-Avila, "SHOP and M-SHOP: Planning with Ordered Task Decomposition," Available: https://drum.umd.edu/dspace/handle/1903/517?mode=simple. [Accessed: 6 April 2006].

[50] Marco Pistore, Paolo Traverso and Piergiorgio Bertoli, "Planning and Monitoring Web Service Composition," presented at AIMSA 2004, Varna, Bulgaria, 2004.

[51] Marco Pistore, Paolo Traverso, and Piergiorgio Bertoli, "Automated Composition of Web Services by Planning in Asynchronous Domains", in *Proceedings of the 15th International Conference on Automated Planning and Scheduling*, 2005, pp. 2-11.

[52] Michele Trainotti, Marco Pistore, Fabio Barbon, Piergiorgio Bertoli, Annapaola Marconi, Paolo Traverso and Gabriele Zacco, "ASTRO: Supporting Web Service Development by Automated Composition, Monitoring and Verfication", in *Proceedings of the 16th International Conference on Automated Planning and Scheduling (Software Demonstration)*, 2006, pp. 28-31.

[53] Michele Trainotti, Marco Pistorem, Gaetano Calabrese, Gabriele Zacco, Gigi Lucchese, Fabio Barbon, Piergiorgio Bertoli and Paolo Traverso, "ASTRO: Supporting Composition and Execution of Web Services", in *Proceedings of the ICSOC 2005*, 2005, pp. 495-501.