WEB SERVICE COMPOSITION UNDER RESOURCE ALLOCATION CONSTRAINTS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ERMAN KARAKOÇ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

APRIL 2007

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Volkan Atalay
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Pınar Şenkul
Supervisor

Examining Committee Members

Prof. Dr. Adnan Yazıcı (METU, CENG) _____

Asst. Prof. Dr. Pınar Şenkul (METU, CENG) _____

Assoc. Prof. Dr. Ali Doğru (METU, CENG) _____

Assoc. Prof.Dr. İsmail Hakkı Toroslu (METU, CENG) _____

Asst. Prof. Dr. Hürevren Kılıç (Atılım Uni., CENG) _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Erman Karakoç

Signature :

ABSTRACT

WEB SERVICE COMPOSITION UNDER RESOURCE ALLOCATION CONSTRAINTS

Karakoç, Erman

M.S., Department of Computer Engineering

Supervisor : Asst. Prof. Dr. Pinar Senkul

April 2007, 113 pages

Web service composition is an inevitable aspect of web services technology, which solves complex problems by combining available basic services and ordering them to best suit the problem requirements. Automatic composition gives us flexibility of selecting best candidate services at composition time, this would require the user to define the resource allocation constraints for selecting and composing candidate web services. Resource allocation constraints define restrictions on how to allocate resources, and scheduling under resource allocation constraints to provide proper resource allocation to tasks. In this work, web service composition system named as CWSF (Composite Web Service Framework) constructed for users to define a workflow system in which a rich set of constraints can be defined on web services. On the contrary many technologies and studies, CWSF provides a user-friendly environment for modeling web service composition system. The output of the framework is the scheduling of web service composition in which how and when web services are executed are defined. With this work, a language, CWSL is defined to describe workflow, resource allocation constraints, selection and discovery rules of web services and associated semantic information. An important property of CWSF system is converting web service composition problem into a

constraint satisfaction problem to find the best solution that meet the all criteria defined by user. Furthermore, CWSF has ability to display other possible solutions to provides users flexibility. This study also includes semantic matching and mapping facilities for service discovery.

Keywords: Dynamic Web Service Composition, Resource Selection, Constraint Satisfaction Problem, Semantic Web Services

ÖZ

KAYNAK AYRIM KISITLARI ALTINDA WEB SERVİS BİRLEŞİMİ

Karakoç, Erman
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Yrd. Doç. Dr. Pınar Şenkul

Nisan 2007, 113 sayfa

Günümüzün karışık problemlerini sadece bir web servis tarafından çözmek mümkün olamamaktadır, bu yüzden web servis birleşimi web servis teknolojisinin zorunlu bir özelliği haline gelmiştir. Web servis birleşimi temel olarak çalışabilir ve uyumlu web servislerini biraraya getirip sıralayarak problemin tüm gereksinimlerini karşılamayı hedefler. Servislerin otomatik birleşimi kullanıcılara sağladığı esneklik ile birleşim zamanında problemin çözümünde en iyi web servisleri seçmeyi amaçlar. Otomatik birleşim sırasında kullanıcılar servis seçimi ve birleşimi için kaynak ayrım kısıtları belirterek aday web servislerin nasıl seçileceğini tanımlayabilirler. Kaynak ayrım kısıtları, kaynakların dağılımına dair sınırlamalar tanımlar ve bu kısıtlar altında yapılan birleşimin planlanmasını, web servisler için doğru kaynak atamalarının yapılmasını sağlar. Web servis birleşim problemi iş akışı mantığında tanımlanabilir; iş akışı karmaşık işlemleri oluşturmak üzere biraraya gelmiş görevler topluluğudur. Bu çalışmada, CWSF (Composite Web Service Framework) ismiyle web servis birleşim sistemi tanımlanarak kullanıcıların bir iş akışı sistemi üzerinde geniş bir yelpazede web servis kaynakları üzerinde kısıtlamalar koyarak en iyi birleşimi yakalama amaçlanmaktadır. Aynı zamanda birçok mevcut çalışmaların askine kullanımı kolay bir birleşim ortamı sağlayarak kullanıcılara esneklik sağlanması hedeflenmektedir. Sistemin çıktısı web servis birleşim planı olup bu planda servislerin hangi zamanda ve nasıl çalıştırılacağı

belirtilmektedir. Bu çalışma ile CWSL ismiyle bir dil tanımlanarak kullanıcıların iş akışını, kısıtlarını, web servislerin seçimini ve anlamsal bilgilerini tanımlayabileceği bir dil oluşturulmuştur. CWSF sisteminin önemli bir özelliği ise web servis birleşim problemini kısıt problemine dönüştürerek en iyi çözümü bulmayı amaçlamasıdır. Sistem kullanıcılara alternatif çözümler göstererek esneklik sağlayabilmektedir. Aynı zamanda bu çalışma içerisinde anlamsal eşleme ve adresleme ile servislerin keşfedilmesi ve eşleştirmesine imkan sağlanmıştır.

Anahtar Kelimeler: Dinamik Web Servis birleşimi, Kaynak Atama, Kısıt Problemi, Anlamsal Web Servisler

To My Father

ACKNOWLEDGMENTS

Firstly, I would like to thank to my thesis advisor, Asst. Prof. Dr. Pınar Şenkul, for her guidance, support and motivation she provided throughout my research. I also very much like to thank to my mother, father, and my older brother for always caring and for their support. Thanks are also to my girl friend for understanding me during my thesis study.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In recent years, the growing numbers of Web Services have emerged as the Internet develops at a fast rate. The web is now evolving into a distributed device of computation from a collection of information resources [4]. The need for composing the existing Web Services into more complex services is also increasing, since it can result into new and more useful solutions. Currently, users either utilize some services that they know already [3], and present service composition research aims at reducing the complexity and time needed to generate and execute a composition and improving its efficiency by selecting the best possible services available at design time. Thus, the ability to efficiently and effectively select and integrate inter-organizational and heterogeneous services on the Web at runtime is an important step towards the development of the Web service applications [14].

Web services are modular, self-describing, self-contained applications that are accessible over the Internet [7]. A composite web service [1] refers to a process consisting of collaborating, heterogeneous web services. The process can be either static or dynamic. In dynamic composition, the web services to be used for activities are selected at run-time by the help of service registries such as UDDI [3] according to the requirements provided by the user. There are several works on web service composition such as [1, 2, 8, 10, 11, 15, 18, 19]. In this work, a web service composition system named as CWSF (Composite Web Service Framework) presented whose main emphasized properties are the ability to express and solve a rich set of constraints on the composite service based on a workflow system, semantic matching and mapping functionalities for service discovery, a user-friendly modeling interface with guidance service.

In this thesis, CWSF is presented where resource allocation management techniques are employed for service selection. Resource management has been recognized as an important aspect of process modeling and scheduling [16, 17]. Web service composition problem is modeled as a constraints satisfaction problem (CSP) and composition plans are generated by utilizing a constraint solver. This method is not applied by current standards or methods for web service composition planning. The output of the framework is the scheduling of web service composition in which how and when web services executed are defined. With this work, a language, CWSL (Composite Web Service Language), is defined to describe workflow, resource allocation constraints, selection and discovery rules of web services and associated semantic information. In this work, semantic structures such as Semantic Inference and Semantic Matching/Mapping is not in the scope of this thesis.

Most of the previous standards or methods on web service composition concentrated on designing run-time behavior of the composition based on the pre-defined rules. However, in these methods, workflow is totally defined; web services are available directly by URL. Furthermore, it is assumed that correct design and web services are available. Therefore, when composition started to execute, composition can not achieve the goal because of wrong design of composition or violation of constraints or non-available web services. Therefore execution of composition might roll-back, or composition can just fail. This wastes not only the web service resources but also development time. Hence, this framework's aim is to find executable schedules by designing and verifying composition using workflow structure, resource allocation constraints and semantic structures without trying possible executions. Mainly, the power of CWSL comes with the dynamism of defining and planning of composition. Main contributions of CWSF can be defined as;

- A workflow language supporting nested block structure (Sequence, And, Or, Xor, Iteration, Condition, Decision, Web Service Template Blocks). This type of a nested block structure with many block structures does not exist in many standards or methods

- A rich set of resource allocation constraints on composition including cost, control, ordering, logical constraints and business rules. The power of such a constraint definition library does not exist in current methodologies.
- Semantic Discovery of Web Services using UDDI and Query Structures for plan generation
- Converting the Web Service Composition Problem to Constraint Satisfaction Problem using a developed Constraint Translator.
- The user-friendly CWSL Designer tool is constructed for defining CWSL representations, displaying alternative solutions, and a guiding system to orient users.
- CWSF displays other possible solutions to provide users flexibility in selection of other executable schedules. This approach does not exist in current standards.

Generally, the automation means that either the method can generate the process model automatically, or the method can locate the correct services if an abstract process model is given [15]. CWSF encapsulates these two methods, it helps to define abstract process model by the guiding system, and locates the correct services based on defined process model using constraints.

Organization of this thesis is as follows: Chapter 2 presents the previous and related work, current standards and technologies. In Chapter 3, Composite Web Service Framework (CWSF) is presented. Chapter 4 defines the structure and simple usage of CWSL Designer. Chapter 5 concludes the thesis.

CHAPTER 2

PREVIOUS WORK

2.1 Web Service Model

A web service is a software system identified by a URL (Uniform Resource Locator), whose public interfaces and bindings are defined and described using XML (Extensible Markup Language). Its definition can be discovered by other software systems. These systems may then interact with the web service in a manner prescribed by its definition, using XML-based messages conveyed by internet protocols. This definition has been published by the World Wide Web consortium W3C, in the Web Services Architecture document [40].

The web service model consists of three entities, the service provider, the service broker (service registry) and the service consumer. Figure 2.1 [48] shows a graphical representation of the traditional web service model:

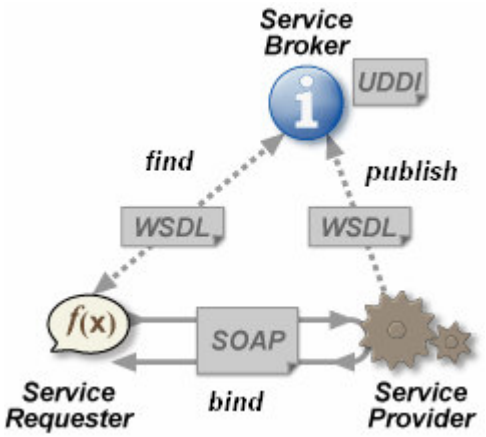

Figure 2.1 Web Service Model

The service provider creates or simply offers the web service. The service provider needs to describe the web service in a standard format, which is XML and publish it in a central Service Registry. The service registry contains additional information about the service provider, such as address and contact information of the providing company, and technical details about the service. The Service Consumer retrieves the information from the registry and uses the service description obtained to bind to and invoke the web service [48]. The appropriate methods are depicted in Figure 2.1 by the keywords 'publish', 'bind' and 'find'. In order to achieve communication among applications running on different platforms and written in different programming languages, standards are needed for each of these operations. Web services architecture is loosely coupled and service oriented. The Web Service Description Language WSDL [14] uses the XML format to describe the methods provided by a web service, including input and output parameters, data types and the transport protocol, which is typically HTTP, to be used. The Universal Description Discovery and Integration (UDDI) standard [3] suggests means to publish details about a service provider, the services that are stored and the opportunity for service consumers to find service providers and web service details. The Simple Object Access Protocol (SOAP) [21] is used for XML formatted information exchange among the entities involved in the web service model.

In the following subsections, brief information about web service standards such as WSDL, SOAP, and UDDI are provided.

2.1.1 WSDL

WSDL (Web Service Description Language) [14] addresses standardized and structured communication protocols and message formats by defining an XML grammar for describing network services as collections of communication endpoints capable of exchanging messages. WSDL service definitions provide documentation for distributed systems and serve as a recipe for automating the details involved in applications communication.

A WSDL document defines services as collections of network endpoints, or ports. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: messages, which are abstract descriptions of the data being exchanged, and port types which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitute a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service. Hence, a WSDL document uses the following elements:

- Types– a container for data type definitions using some type system (such as XSD).
- Message– an abstract, typed definition of the data being communicated.
- Operation– an abstract description of an action supported by the service.
- Port Type–an abstract set of operations supported by one or more endpoints.
- Binding– a concrete protocol and data format specification for a particular port type.
- Port– a single endpoint defined as a combination of a binding and a network address.
- Service– a collection of related endpoints.

2.1.2 SOAP

Simple Object Access Protocol (SOAP) [21] is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics [59].

SOAP is a protocol for exchanging XML-based messages over computer network, normally using HTTP. SOAP forms the foundation layer of the Web services stack, providing a basic messaging framework that more abstract layer can build on.

Here is an example of SOAP message of a web service

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetails xmlns="http://warehouse.example.com/ws">
       <getProductDetailsResult>
         <productName>Toptimate 3-Piece Set</productName>
         <productID>827635</productID>
         <description>3-Piece luggage set.</description>
         <price>96.50</price>
         <inStock>true</inStock>
       </getProductDetailsResult>
     </getProductDetails>
   </soap:Body>
 </soap:Envelope>
```

2.1.3 UDDI

UDDI [3] (Universal Description, Discovery, and Integration) is a platform-independent, XML-based registry for businesses worldwide to list themselves on the Internet. UDDI is an open industry initiative, sponsored by OASIS [63], enabling businesses to publish service listings and discover each other and define how the services or software applications interact over the Internet. Based on UDDI specifications, web service providers publish themselves, and web service consumers can query services.

2.1.4 Other Standards

In the distributed environment of web service execution, many standards or protocols exist to manage web service composition. Two of them are WS-Security that defines how to use XML Encryption and XML Signature in SOAP to secure message exchanges, and WS-ReliableMessaging that is a protocol for reliable messaging between two Web services. Complete list of web service specifications can be found at [37].

2.2 The need for web service composition

Web services are considered as self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Nowadays, an increasing amount of companies and organizations implement their core business and outsource other application services over Internet. Thus, the ability to

efficiently and effectively select and integrate inter-organizational and heterogeneous services on the Web at runtime is considered to be an important step towards the development of the Web service applications. In particular, if no single Web service can satisfy the functionality required by the user, there should be a possibility to combine existing services together in order to fulfill the request. This trend has triggered a considerable number of research efforts on the composition of Web services both in academia and in industry [15].

The basic web services infrastructure suffices to implement simple interactions between a client and a web service. If the implementation of a web service's business requirement involves the invocation of other web services, it is necessary to combine the functionality of several web services. In this case, we speak of a composite service [41]. Web service composition lets developers create applications on top of service oriented computing's native description, discovery, and communication capabilities. Such applications are rapidly deployable and offer developers reuse possibilities and users seamless access to a variety of complex services [36].

Since it is a widely used approach to use conventional programming languages to link components to a composite web service, and thereby bridge heterogeneous middleware platforms, it becomes necessary to develop a Service Composition Middleware to support composition in terms of abstractions and infrastructure as well [36]. Programming languages focus on APIs rather than on the actual business logic. Different approaches and the need for workflow modeling have finally led to the development of the business process execution language for web services BPEL4WS [25]. A composition model and language to specify the services involved in the composition, a development environment with a graphical user interface to drag and drop web service components and a run-time environment to execute the business logic can be identified as the three main elements of a web services composition middleware. A service composition middleware requires the web services to be precisely described in their functionality, interfaces and protocols they support. Conventional middleware lacks exactly those features.

2.3 Problems of Web Service Composition

Despite all efforts, the Web service composition still is a highly complex task, and it is already beyond the human capability to deal with the whole process manually. The complexity, in general, comes from the following sources.

- The number of services available over the Web increased dramatically during the recent years, and one can expect to have a huge Web service repository to be searched.
- Web services can be created and updated on the fly, thus the composition system needs to detect the updating at runtime and the decision should be made based on the up to date information.
- Web services can be developed by different organizations, which use different concept models to describe the services, however, a unique structure does not exist to define and evaluate the Web services in an identical means.
- On web service composition, there are many candidates for a web service, and it is necessary to select the correct web service to achieve the composition goal. Finding and integrating a web service without making any error has importance on execution.
- There can be many resource allocation constraints when selecting web services or achieving composition constraints in a dynamic composition that defines how web services will be selected.

Therefore, an automated or semi automated tool is critical when building composite web services. Although the different methods provide different levels of automation in service composition, it is not possible to say the higher automation the better. Because of the complexity of web service environment, generation of composite web services in an automatic way is not possible.

In addition, currently, standardization works of service composition do not cover definitions of all key requirements that every composition approach must satisfy (such as scalability, dependability, and correctness) [36].

2.4 Standards for Web Service Composition

Currently BPEL and OWL-S are the two composition languages that are standardized and widely used in community.

2.4.1 BPEL

Business Process Execution Language [5] is an XML based language that supports process oriented service composition. Developed by BEA, IBM, Microsoft, SAP, and Siebel, BPEL is currently being standardized by the Organization for the Advancement of Structured Information Standards.

BPEL composition interacts with a Web services subset to achieve a given task. In BPEL, the composition result is called a process, participating services are partners, and message exchange or intermediate result transformation is called an activity. A process thus consists of a set of activities. A process interacts with external partner services through a WSDL interface [36].

To define a process, BPEL source file (.bpel) is required to describe activities; and a process interface (.wsdl), which describes ports of a composed service; and an optional deployment descriptor (.xml), which contains the partner services' physical locations.

BPEL has several element groups, but the basic ones are;

- process initiation: <process>
- definition of services participating in composition: <partnerLink>
- synchronous and asynchronous calls: <invoke>, <invoke>... <receive>
- intermediate variables and results manipulation: <variable>, <assign>, <copy>
- error handling: <scope>, <faultHandlers>
- sequential and parallel execution: <sequence>, <flow>
- logic control: <switch>

In the following example [36], model of BPEL composition of three services are illustrated. Service A is called synchronously and starts a process. Two asynchronous services, B and C, are then called in parallel using Service A's output as their input. The process waits for their completion and then makes a decision based on the results.

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="test">
    <partnerLinks>
             <partnerLink name="client"/>
             <partnerLink name="serviceA"/>
             <partnerLink name="serviceB"/>
             <partnerLink name="serviceC"/>
      </partnerLinks>
      <variables>
             <variable name="processInput"/>
             <variable name="AInput"/>
             <variable name="AOutput"/>
             <variable name="BCInput"/>
             <variable name="BOutput"/>
             <variable name="COutput"/>
             <variable name="processOutput"/>
             <variable name="AError"/>
        </variables>
        <sequence>
                <receive name="receiveInput" variable="input"/>
                    <assign><copy>
                        <from variable="processInput"/><to
variable="AInput"/>
                    </copy></assign>
                    <scope>
                        <faultHandlers>
                            <catch faultName="faultA" fault-
Variable="AError"/>
                         </faultHandlers>
                         <sequence>
                             <invoke name="invokeA" partner-
Link="serviceA"
                                    inputVariable="AInput" output-
                           Variable="AOutput"/>
                            </sequence>
                    </scope>
                    <assign><copy>
                        <from variable="AOutput"/><to variable="BCInput"/>
                    </copy></assign>
                    <flow>
                        <sequence>
                            <invoke name="invokeB"
                              partner-Link="serviceB"
                           inputVariable="BCInput"/>
                            <receive name="receive_invokeB"
                              partnerLink="serviceB" variable="BOutput"/>
                        </sequence>
                        <sequence>
                            <invoke name="invokeC" partner-Link="serviceC"
                                 inputVariable="BCInput"/>
                           <receive name="receive_invokeC"
```

```
                                    partnerLink="serviceC"
                                variable="COutput"/>
                            </sequence>
                        </flow>
                        <switch><case>
                            <!– assign value to processOutput –>
                        </case></switch>
                        <invoke name="reply" partnerLink="client"
                           inputVariable="processOutput"/>
                    </sequence>
</process>
```

There are several BPEL orchestration server implementations for both J2EE and .NET platforms, including IBM WebSphere [42], Oracle BPEL Process Manager [43], Microsoft BizTalk 2004 [44], and Active BPEL [45]

2.4.2 OWL-S

The Web Ontology Language (OWL) [60] is a markup language for publishing and sharing data using ontologies on the World Wide Web. OWL is designed for use by applications that need to process the content of information instead of just presenting information to humans. It facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics. OWL has three increasingly expressive sublanguages: OWL Lite, OWL DL, and OWL Full. OWL is based on earlier languages OIL and DAML+OIL, and is a W3C recommendation.

The Semantic Web vision is to make Web resources accessible by content as well as by keywords. Web services play an important role in this: users and software agents should be able to discover, compose, and invoke content using complex services. The DARPA Agent Markup Language (DAML) extends XML and the Resource Description Framework (RDF) to provide a set of constructs for creating machine-readable ontologies and markup information. The DAML program's Semantic Web contribution is the Web Ontology Language for Services (www.daml.org/services). OWL-S (previously known as DAML-S) is a services ontology that enables automatic service discovery, invocation, composition, interoperation, and execution monitoring [36]. OWL-S models services using three-part ontology:

- a service profile describes what the service requires from users and what it gives to them;
- a service model specifies how the service works;
- a service grounding gives information on how to use the service.

The process model is a service model subclass that describes a service in terms of inputs, outputs, preconditions, post conditions, and its own sub processes. All information such as input and output of services is defined as OWL ontologies. In the process model, we can describe composite processes and their dependencies and interactions. OWL-S distinguishes three types of processes: atomic, which have no sub processes; simple, which are not directly invocable and are used as an abstraction element for either atomic or composite processes; and composite, which consist of sub processes. Constituent processes are specified using flow-control constructs: sequence, split, split+join, unordered, choice, if-then-else, iterate, and repeat-until [36].

In the following OWL-S example, a single THY Plane Web Service is used in the AtomicProcess of OWL-S. Process Input and Output Ontologies are given and necessary ground information is available to access the web service. XSL conversions exist to define how web service input/output of THY Web Service will be converted to input/output of OWL-S composition.

```
....
....
<owl:Ontology rdf:about="">
        <owl:imports rdf:resource="&service;"/>
        <owl:imports rdf:resource="&profile;"/>
        <owl:imports rdf:resource="&process;"/>
        <owl:imports rdf:resource="&grounding;"/>
      <owl:imports rdf:resource="&concepts;"/>
</owl:Ontology>

<!-- Service description -->
<service:Service rdf:ID="MyTHYQueryServiceService">
        <service:presents rdf:resource="#MyTHYQueryServiceProfile"/>

        <service:describedBy rdf:resource="#MyTHYQueryServiceProcess"/>

        <service:supports rdf:resource="#MyTHYQueryServiceGrounding"/>
</service:Service>

<!-- Profile description -->
<profile:Profile rdf:ID="MyTHYQueryServiceProfile">
        <service:isPresentedBy rdf:resource="#MyTHYQueryServiceService"/>

        <profile:serviceName xml:lang="en">MyTHYQueryService</profile:serviceName>
```

```
        <profile:textDescription xml:lang="en">THY
QueryService</profile:textDescription>

        <profile:hasInput rdf:resource="#inMyTHYQueryService"/>

        <profile:hasOutput rdf:resource="#outMyTHYQueryService"/>
</profile:Profile>

<!-- Process description -->
<process:AtomicProcess rdf:ID="MyTHYQueryServiceProcess">
        <service:describes rdf:resource="#MyTHYQueryServiceService"/>

        <process:hasInput rdf:resource="#inMyTHYQueryService"/>

        <process:hasOutput rdf:resource="#outMyTHYQueryService"/>
</process:AtomicProcess>

<process:Input rdf:ID="inMyTHYQueryService">
        <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/QueryCriteriaForService.o
wl#QueryCriteriaForService</process:parameterType>
        <rdfs:label>MyTHYQueryService input</rdfs:label>
</process:Input>

<process:Output rdf:ID="outMyTHYQueryService">
        <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/FlighReservationInfo.owl#
FlighReservationInfo</process:parameterType>
        <rdfs:label>MyTHYQueryService output</rdfs:label>
</process:Output>

<!-- Grounding description -->
<grounding:WsdlGrounding rdf:ID="MyTHYQueryServiceGrounding">
        <service:supportedBy rdf:resource="#MyTHYQueryServiceService"/>
        <grounding:hasAtomicProcessGrounding
rdf:resource="#MyTHYQueryServiceProcessGrounding"/>
</grounding:WsdlGrounding>

<grounding:WsdlAtomicProcessGrounding rdf:ID="MyTHYQueryServiceProcessGrounding">
        <grounding:owlsProcess rdf:resource="#MyTHYQueryServiceProcess"/>
        <grounding:wsdlDocument
rdf:datatype="&xsd;#anyURI">http://localhost:8083/WebServiceTestModule/services/THYQ
ueryService?wsdl</grounding:wsdlDocument>
        <grounding:wsdlOperation>
                <grounding:WsdlOperationRef>
                        <grounding:portType
rdf:datatype="&xsd;#anyURI">http://localhost:8083/WebServiceTestModule/services/THYQ
ueryService?wsdl#THYQueryService</grounding:portType>
                        <grounding:operation
rdf:datatype="&xsd;#anyURI">http://localhost:8083/WebServiceTestModule/services/THYQ
ueryService?wsdl#query</grounding:operation>
                </grounding:WsdlOperationRef>
        </grounding:wsdlOperation>

        <grounding:wsdlInputMessage
rdf:datatype="&xsd;#anyURI">http://localhost:8083/WebServiceTestModule/services/THYQ
ueryService?wsdl#queryRequest</grounding:wsdlInputMessage>
        <grounding:wsdlInput>
                <grounding:WsdlInputMessageMap>
                        <grounding:owlsParameter
rdf:resource="#inMyTHYQueryService"/>
                        <grounding:wsdlMessagePart
rdf:datatype="&xsd;#anyURI">http://localhost:8083/WebServiceTestModule/services/THYQ
ueryService?wsdl#criteria</grounding:wsdlMessagePart>
                        <grounding:xsltTransformationString>
                          <![CDATA[
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:concepts="http://localhost:8080/examples/QueryCriteriaForService.owl#">
        <xsl:template match="//concepts:QueryCriteriaForService">
                <QueryCriteriaForService>
                        <names>
```

```
                        <xsl:value-of select="concepts:names"/>
                    </names>
                    <operations>
                        <xsl:value-of select="concepts:operations"/>
                    </operations>
                    <values>
                        <xsl:value-of select="concepts:values"/>
                    </values>
                </QueryCriteriaForService>
        </xsl:template>
</xsl:stylesheet>
                      ]]>
                    </grounding:xsltTransformationString>

                </grounding:WsdlInputMessageMap>
        </grounding:wsdlInput>

        <grounding:wsdlOutputMessage
rdf:datatype="http://www.w3.org/2001/XMLSchemae#anyURI">http://localhost:8083/WebSer
viceTestModule/services/THYQueryService?wsdl#queryResponse</grounding:wsdlOutputMess
age>
        <grounding:wsdlOutput>
                <grounding:WsdlOutputMessageMap>
                    <grounding:owlsParameter
rdf:resource="#outMyTHYQueryService"/>
                    <grounding:wsdlMessagePart
rdf:datatype="&xsd;#anyURI">http://localhost:8083/WebServiceTestModule/services/THYQ
ueryService?wsdl#queryReturn</grounding:wsdlMessagePart>

                    <grounding:xsltTransformationString>
                      <![CDATA[
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:template match="/queryReturn">
                <xsl:variable name="X1" select="destinationDate"/>
                <xsl:variable name="X2" select="destinationPlace"/>
                <xsl:variable name="X3" select="departureDate"/>
                <xsl:variable name="X4" select="departurePlace"/>
                <xsl:variable name="X5" select="cost"/>

                <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:concepts="http://localhost:8080/examples/FlighReservationInfo.owl#">
                    <concepts:FlighReservationInfo>
                        <concepts:destinationDate>
                            <xsl:value-of select="$X1"/>
                        </concepts:destinationDate>
                        <concepts:destinationPlace>
                            <xsl:value-of select="$X2"/>
                        </concepts:destinationPlace>
                        <concepts:departureDate>
                            <xsl:value-of select="$X3"/>
                        </concepts:departureDate>
                        <concepts:departurePlace>
                            <xsl:value-of select="$X4"/>
                        </concepts:departurePlace>
                        <concepts:cost>
                            <xsl:value-of select="$X5"/>
                        </concepts:cost>
                    </concepts:FlighReservationInfo>
                </rdf:RDF>
        </xsl:template>
</xsl:stylesheet>
                      ]]>
                    </grounding:xsltTransformationString>
        </grounding:WsdlOutputMessageMap>
    </grounding:wsdlOutput>

        </grounding:WsdlAtomicProcessGrounding>
</rdf:RDF>
```

Researchers have proposed methods for transferring OWL-S descriptions to Prolog [29] and Petri-net [49] based notation to further analyze verification. In the Prolog approach, the developer manually translates an OWL-S description to Prolog, which makes it possible to find an adequate plan for composing Web services for a target description. That is, for a given pool of available Web services, it is possible to use logical inference rules to automate service allocation for the required task. In the Petri-net approach, an OWL-S description is automatically translated into Petri nets. Developers use this representation to automate tasks such as simulation, validation, verification, composition, and performance analysis [36].

2.5 Web Service Composition Methods

In today's Web, Web services are created and updated on the fly. It is already beyond human ability to analyze them and generate the composition plan manually. A number of approaches have been proposed to tackle that problem. Most of them are inspired by the researches in cross-enterprise workflow and AI planning [15].

The dynamic workflow methods provide the means to bind the abstract nodes with the concrete resources or services automatically. On the other hand, dynamic composition methods are required to generate the plan automatically. Most methods in such category are related to AI planning and deductive theorem proving. The general assumption behind such kind of methods is that each Web service can be specified by its preconditions and effects in the planning context. Firstly, a Web service is a software component that takes the input data and produces the output data. Thus the preconditions and effects are the input and the output parameters of the service respectively. Secondly, the Web service also alters the states of the world after its execution. Therefore, the world state pre-required for the service execution is the precondition, and new states generated after the execution is the effect [15].

2.5.1 General Architecture of Service Composition Frameworks

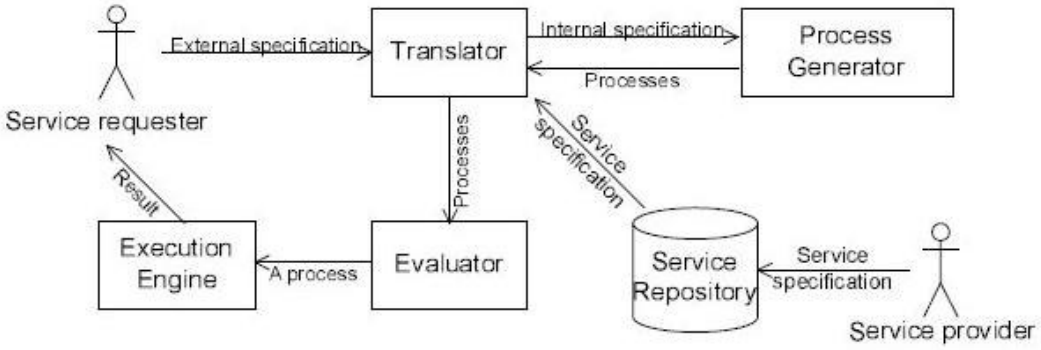

Figure 2.2 General Architecture of Web Service Composition Systems

A general framework of the service composition system is illustrated in Figure 2.2 [15]. The composition system has two kinds of participants, service provider and service requester. The service providers propose Web services for use. The service requesters consume information or services offered by service providers. The system also contains the following components: translator, process generator, evaluator, and execution engine and service repository. The translator translates between the external languages used by the participants and the internal languages used by the process generator. For each request, the process generator tries to generate a plan that composes the available services in the service repository to fulfill the request. If more than one plan is found, the evaluator evaluates all plans and proposes the best one for execution. The execution engine executes the plan and returns the result to the service provider [15].

2.5.2 Workflow Based Frameworks

In the workflow-based composition methods, it is necessary to distinguish the static and dynamic workflow generation. The static one means that the requester should build an abstract process model before the composition planning starts. The abstract process model includes a set of tasks and their data dependency. Each task contains a query clause that is used to search the real atomic Web service to fulfill the task.

In that case, only the selection and binding of atomic Web service is done automatically by program. The most commonly used static method is to specify the process model in graph. On the other hand, the dynamic composition both creates process model and selects atomic services automatically. This requires the requester to specify several constraints, including the dependency of atomic, the user's preference and so on. In this section, some composition methods are explained.

EFlow [15, 27] is a platform for the specification, enactment and management of composite services. EFlow uses a static workflow generation method. A composite service is modeled by a graph that defines the order of execution among the nodes in the process. The graph is created manually but it can be updated dynamically. The graph may include service, decision and event nodes. Service nodes represent the invocation of an atomic or composite service, decision nodes specify the alternatives and rules controlling the execution flow, and event nodes enable service processes to send and receive several types of events. Arcs in the graph denote the execution dependency among the nodes. Although the graph should be specified manually, EFlow provides the automation to bind the nodes with concrete services. The definition of a service node contains a search recipe that can be used to query actual service either at process instantiation time or at runtime.

In another composition work, the authors [15, 26] further refine the service composition platform and propose a prototype of composite service definition language (CSDL). An interesting feature of CSDL is that it distinguishes between invocation of services and operations within a service. It provides the adaptive and dynamic features to cope with the rapidly evolving business and IT environment in which Web services are executed.

2.5.3 AI Planning Based Frameworks

Many research efforts tackling Web service composition problem via AI planning have been reported [15]. In general, a planning problem can be described as a five tuple $<S, S_0, G, A, \Gamma>$, where S is the set of all possible states of the world, $S_0 \subset S$ denotes the initial state of the world, $G \subset S$ denotes the goal state of the world the

planning system attempts to reach, A is the set of actions the planner can perform in attempting to change one state to another state in the world, and the translation relation $\Gamma \subseteq S \times A \times S$ defines the precondition and effects for the execution of each action. In the terms of Web services, S_0 and G are the initial state and the goal state specified in the requirement of Web service requesters. A is a set of available services. Γ further denotes the state change function of each service.

McIlraith et. al. [15, 29, 30, 31] adapt and extend the Golog language for automatic construction of Web services. Golog is a logic programming language built on top of the situation calculus. The authors address the Web service composition problem through the provision of high-level generic procedures and customizing constraints. Golog is adopted as a natural formalism for representing and reasoning about this problem.

The general idea about this method is that software agents could reason about Web services to perform automatic Web service discovery, execution, composition and inter-operation. The user's request (generic procedure) and constraints can be presented by the first-order language of the situation calculus (a logical language for reasoning about action and change). The authors conceive each Web service as an action - either a PrimitiveAction or a ComplexAction. Primitive actions are conceived as either world-altering actions that change the state of the world or information-gathering actions that change the agent's state of knowledge. Complex actions are compositions of individual actions. The agent knowledge base provides a logical encoding of the preconditions and effects of the Web service actions in the language of the situation calculus. The agents use procedural programming language constructs composed with concepts defined for the services and constraints using deductive machinery. A composite service is a set of atomic services connected by procedural programming language constructs (if-then-else, while and so forth). The authors also propose a way to customize Golog programs by incorporating the service requester's constraints

A strong interest to Web service composition from AI planning community could be explained roughly by similarity between DAML-S and PDDL representations. PDDL is widely recognized as a standardized input for state-of-the-art planners. Moreover, since DAML-S has been strongly influenced by PDDL language, mapping from one representation to another is straightforward (as long as only declarative information is considered). When planning for service composition is needed, DAML-S descriptions could be translated to PDDL format. Then different planners could be exploited for further service synthesis [15].

Medjahed [15, 26] present a technique to generate composite services from high-level declarative description. The method uses composability rules to determine whether two services are composable. The composition approach consists of four phases. First, the specification phase enables high-level description of the desired compositions using a language called Composite Service Specification Language(CSSL). Second, the matchmaking phase uses composability rules to generate composition plans that conform to service requester's specifications. The third phase is selection phase. If more than one plan is generated, in the selection phase, the service requester selects a plan based on quality of composition (QoC) parameters (e.g. rank, cost, etc.). The final phase is the generation phase. A detailed description of the composite service is automatically generated and presented to the service requester.

In METEOR-S [64], they present an approach for achieving constraint driven Web service composition. In this framework, abstract process designer, constraint analyzer, optimizer and binder modules are available. They extend the workflow QoS model to allow for global optimization and composition of Web processes. The process constraints are directly converted to constraints for using Integer Linear Programming Solver.

SWORD [15, 24] is another developer toolkit for building composite Web services using rule-based plan generation. SWORD does not deploy the emerging service-description standards such as WSDL and DAML-S, instead, it uses Entity-Relation

(ER) model to specify the Web services. In SWORD, a service is modeled by its preconditions and post conditions. They are specified in a world model that consists of entities and relationships among entities. A Web service is represented in the form of a Horn rule that denotes the post conditions are achieved if the preconditions are true. To create a composite service, the service requester only needs to specify the initial and final states for the composite service, and then the plan generation can be achieved using a rule-based expert system.

Some other AI planning techniques are proposed for the automatic composition of Web services. In [33] the SHOP2 planner is applied for automatic composition of Web services, which are provided with DAML-S descriptions. SHOP2 is a Hierarchical Task Network (HTN) planner. The authors believe that the concept of task decomposition in HTN planning is very similar to the concept of composite process decomposition in DAML-S process ontology. The authors also claim that the HTN planner is more efficient than other planning language, such as Golog. In their paper, the authors give a very detailed description on the process of translating DAML-S to SHOP2. In particular, most control constructs can be expressed by SHOP2 in an explicit way [15].

Sirin et al [19, 32] present a semi-automatic method for web service composition. Each time when a user has to select a Web service, all possible services that match with the selected service is presented to the user. The choice of the possible services is based both on functionalities and non-functional attributes. The functionalities (parameters) are presented by OWL classes and OWL reasoner is applied to match the services. A match is defined between two services that an output parameter of one service is the same OWL class or subclass of an input parameter of another service. The OWL inference engine can order the matched services so that the priority of the matches is lowered when the distance between the two types in the ontology tree increases. If more than one match is found, the system filters the services based on the non-functional attributes that are specified by the user as constraints. Only those services that pass the non-functional constraints can be presented to the service requester.

2.6 Constraint Satisfaction Problem

CWSF system converts web service composition problem into a constraint satisfaction problem to find executable solutions. Therefore, it is necessary to understand and review the basic concepts of Constraint Satisfaction Problem (CSP).

2.6.1 Introduction to CSP

Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it. [34]

Fast increasing computing power in the 1960s led to a wealth of works around problem solving, at the root of Operational Research, Numerical Analysis, Symbolic Computing, Scientific Computing, and a large part of Artificial Intelligence and programming languages. Constraint Programming is a discipline that gathers, interbreeds, and unifies ideas shared by all these domains to tackle decision support problems [35].

Constraint programming has been successfully applied in numerous domains. Recent applications include computer graphics (to express geometric coherence in the case of scene analysis), natural language processing (construction of efficient parsers), database systems (to ensure and/or restore consistency of the data), operations research problems (like optimization problems), molecular biology (DNA sequencing), business applications (option trading), electrical engineering (to locate faults), circuit design (to compute layouts), etc. Current research in this area deals with various foundational issues, with implementation aspects and with new applications of constraint programming [35].

2.6.2 Definition

A constraint is simply a logical relation among several unknowns (or variables), each taking a value in a given domain [35]. A constraint thus restricts the possible values that variables can take; it represents some partial information about the variables of interest.

For instance, the circle is inside the square relates two objects without precisely specifying their positions, i.e., their coordinates. Now, one may move the square or the circle and he or she is still able to maintain the relation between these two objects. Also, one may want to add other object, say triangle, and introduce another constraint, say square is to the left of the triangle. From the user (human) point of view, everything remains absolutely transparent.

2.6.3 Constraint Programming

As stated in [35], constraint programming is the study of computational systems based on constraints. The idea of constraint programming is to solve problems by stating constraints (conditions, properties) which must be satisfied by the solution.

The formulation and the resolution of combinatorial problems are the two main goals of the constraint programming domain. This is an essential way to solve many interesting industrial problems such as scheduling, planning or design of timetables. The main interest of constraint programming is to propose to the user to model his problem without being interested in the way the problem is solved.

Constraint programming allows solving combinatorial problems modelized by a constraint satisfaction problem (CSP). Formally, a CSP is defined by a triplet (X, D, C) such as:

Variables:

$X = \{X_1, X_2,..., X_n\}$ is the set of variables of the problem.

Domain:

D is a function which associates to each variable X_i its domain $D(X_i)$, i.e., the possible values that can be affected to X_i.

Constraints:

$C = \{C1, C2,..., Ck\}$ is the set of constraints. Each constraint C_j is a relation between a subset of variables which restricts the domain of each one.

A constraint is satisfied if the tuple of the values of its variables belongs to the relation describing the constraint. Thus, solving a CSP consists in finding a tuple on the set of variables such that each constraint is satisfied. Moreover, several types of variables can be distinguished:

- Integer variables are variables described on domains containing integer values.
- Set variables are variables described on domains containing sets of values.
- Real variables are variables described on continuous domains and generally use intervals to represent values.

CHAPTER 3

COMPOSITE WEB SERVICE FRAMEWORK

In this chapter, we present a web service composition system architecture named as Composite Web Service Framework (CWSF) that provides modules to model web services in workflow structure and resource allocation constraints to find executable schedules fulfilling these constraints. In order to find schedules, constraint programming approach is used. That is, web service composition scheduler incorporates an off-the-shelf constraint solver to obtain a feasible schedule for the web service composition satisfying resource allocation temporal/causality constraints. The output of framework is the scheduling of web service composition in which how and when web services executed are defined. Ontology-based semantic information of the services is utilized to discover and match compatible web services.

3.1 General Architecture

The major components of the architecture are CWSL Designer, Composition Planner including Service Matching/Mapping Engine, Service Query Engine, Semantic Inference Engine, Plan Generator with Constraint solver and Constraint translator, UDDI Registry and its semantic wrapper, web services, ontology and context model repositories. Core architecture is presented in Fig. 3.1. Composition properties are defined in the scripted language of composition named as CWSL (Composite Web Service Language).

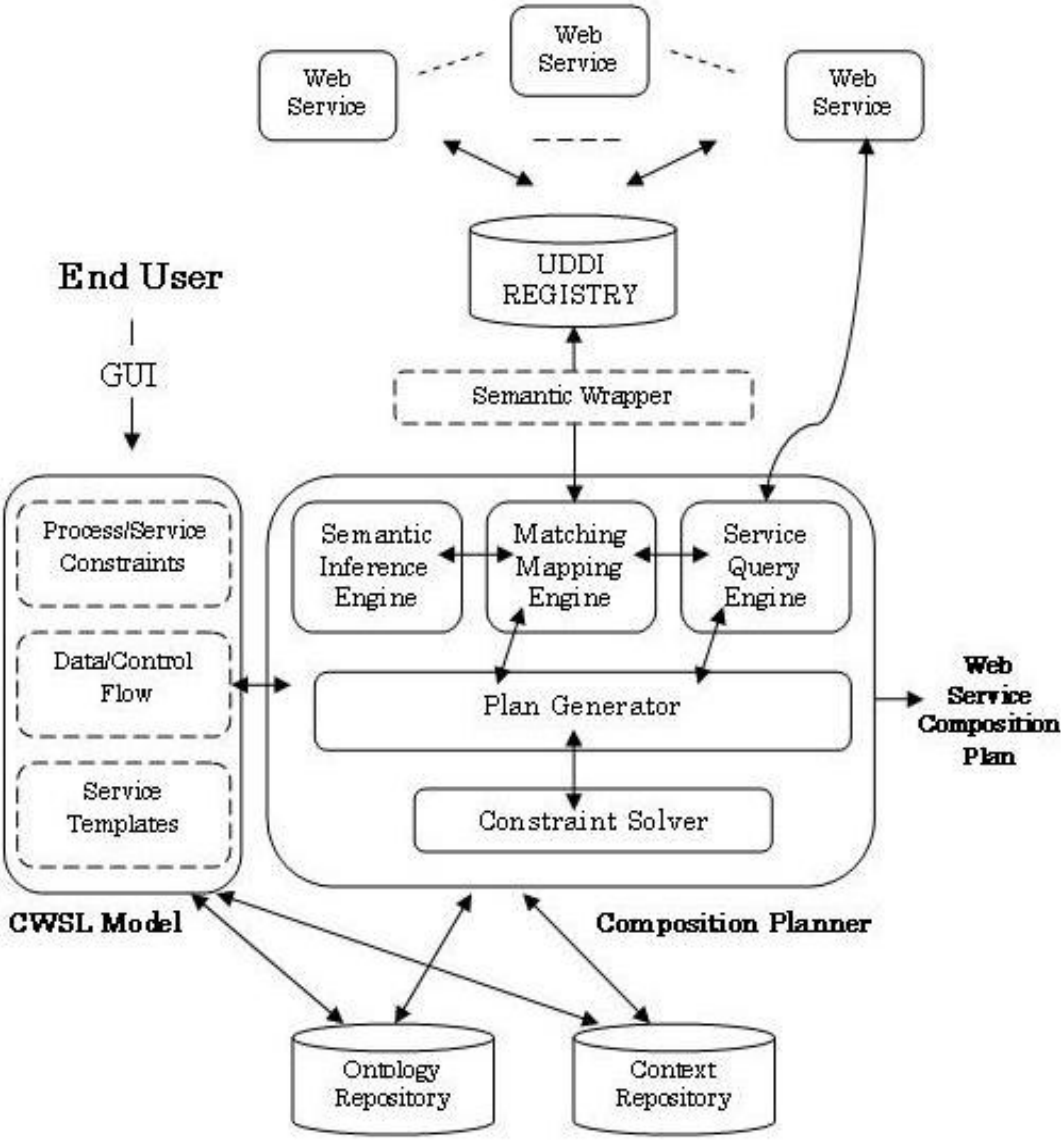

Figure 3.1 General Architecture of CWSF

In the CWSL model, workflow structure of the process, constraints and service templates are defined. Functionality of the required services is expressed by using service templates. A service template defines how an abstract service processes input/output, which parameters it uses, how it can be associated with ontology, which role it has in the process, query structures for selecting web services. Process and service constraints introduce rules as to how a service will be selected from the set provided by the service discovery engine. Plan generator module has overall control of the plan generation phase. Translation of web service composition to constraint solver problems is the responsibility of the plan generator module. This module uses the constraint engine to produce optimal service sets based on constraints to generate plans. Although it is not in fully implemented, it is possible to convert the executable schedule into BPEL target language, since output of CWSF is a subset of constructs of BPEL. Once translation is completed, the resulting process model can be executed by an external BPEL engine. It also is possible to translate the plan into other language such as OWL-S. Semantic Inference Engine and Matching/Mapping Engine is not in the scope of this thesis. In the following subsections details of the architecture is given with emphasis on the modules in the scope of this thesis work.

3.2 Motivation Example – Traveling

An example of a web service composition problem would be a “Travel Planner” system that aggregates and combines multiple web services such as flight booking, travel booking, accommodation booking, car rental, and free activity planning, which are executed sequentially or concurrently based on traveler’s choices.

Traveling example has enough complexity to demonstrate usability and necessity of web service composition because many web services coordinate to achieve a complex goal by combining simpler goals. The process model underlying a composite service identifies the functionalities required by the service templates to be composed and their interactions and dependencies.

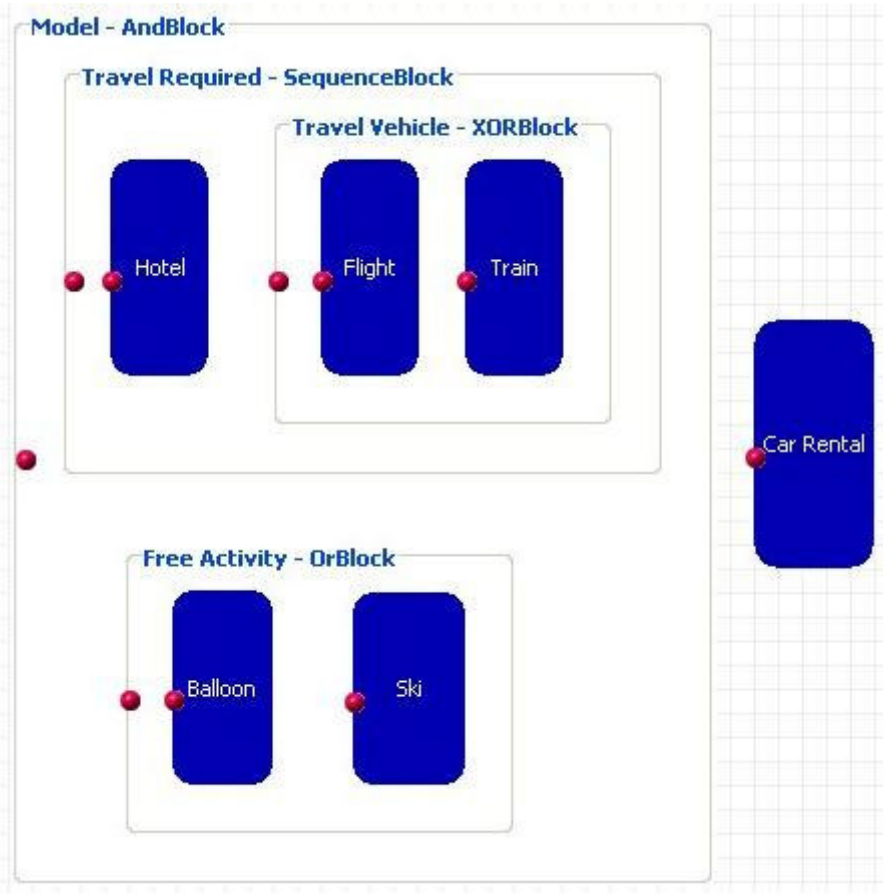

Figure 3.2 Travel Planner Example

Travel planner system example is shown in Figure 3.2. Goal of a travel reservation process is to perfectly organize the travel based on the user's constraints. In the planner, following activities are placed; Hotel Booking, Flight or Train Booking, reserving Balloon or Ski activities, and car rental service. In travel planning, Hotel Booking, Travel Vehicle Reservation and Car Rental activities are mandatory activities that should be executed. Traveler can go to holiday location by plane or by train, therefore only one of them should be selected. In addition, he wants to make some free activities such as flying a balloon, or ski in winter. In this example, an ordinary user can define following constraints;

- My maximum budget is only 1500 $ for my travel.
- I want to go to “Antalya” for my holiday.
- I want to start my travel on Wednesday (25.04.2007) and finish at Sunday (29.04.2007).
- Distance between hotel location and city center should not be more than 15 km.
- Hotel should have minimum four stars.
- I can give maximum 600 $ for 4 days for my hotel.
- I can give extra 200 $ if hotel is “Hilton”.
- I prefer Plane if it is cheaper than Train.
- Free activities are not mandatory; if they are available then I will be happier.
- In either case, I do not want to spend more than 250 $ for my free activities.
- Car Rental is a must for my travel.
- I want to use a “Toyota” Car during my holiday, no other cars.
- Hotel Company and Car Rental Company should be the same.
- I can pay 100$ for car rental,.
- I want to go by plane if it is less than 100 $.
- If I go by train, time of travel should not be more than 10 hours.
- My Balloon tour should last at least one hour.
- Ski activity should include an instructor.
- Also if ski activity is selected, then my ski activity should occur at least 2 times.

In addition to above constraints, implicit time/ordering constraints are defined based on the workflow structure, such as forcing the selection of travel method either plane or train. Car rental service selection should take place after selection of hotel that means car rental can use the output or information of the selected hotel service. Details for the complete CWSL representation about this example are given in Appendix A.1.

There exist many sub-goals in the travel domain, and these goals affect each other as seen in the model and constraints. Main goal of composition should be achieved for successful travel planning. For an ordinary user, following main problems arise;

- Ordinary user can not have knowledge of web services technology, composition properties, how they coordinate and they work constraint engine, semantic structures, etc. He just wants to go for a holiday.
- Ordinary user can think only about sub-goals such as Hotel selection; he can not consider the effect of selection of a web service rather than another web service. Also, this selection affects other sub-goals, and therefore it affects the main-goal. Selection of a single web service can result in fail in all solutions. Therefore, he should consider all process constraints.
- If every web service template in traveling example has ten candidates, then number of possible solutions is the cartesian product of candidate web services. Then number of solutions for the traveling example is 10*10*10*10*10 = 100000; for a human, it is not possible to consider all alternatives and selecting the best solution. Also for many programming approaches, it is not easy to generate and select solutions. Since many constraints can be defined on the traveling example, it is necessary to solve constraints and generate solutions efficiently.

Because of the reasons defined above, it is necessary to develop Composite Web Service Framework (CWSF) under resource allocation constraints to propose the best solution to traveler. Constraint library of the system should have comprehensive capabilities to define user's constraints easily with defining flow information of service templates in a workflow structure.

3.3 Main Contributions of Composite Web Service Framework

Mainly, the power of CWSL comes with the dynamism of defining and planning of composition, main contributions of CWSF can be defined as;

- A workflow language supporting nested block structure (Sequence, And, Or, Xor, Iteration, Condition, Decision, Web Service Template Blocks), process variables, etc. This type of nested block structure with many block structure does not exist in similar methodologies.
- Rich set of resource allocation constraints on composition including cost, control, ordering, logical constraints and business rules. The power of constraint definition library does not exist in current methodologies.
- Effectively finding possible solutions over a large set of possible solutions.
- Semantic Discovery of Web Services using UDDI and Query Structures for plan generation.
- Converting the Web Service Composition Problem to a Constraint Satisfaction Problem using the developed Constraint Translator.
- User-friendly design of web service composition is adapted by the CWSL, as well as Semantic Structures with a guiding system to orient users.
- CWSF displays other possible solutions to provide users flexibility to selection of other executable schedules.

3.4 Composite Web Service Language (CWSL)

A specification language is defined to model web service composition named as CWSL (Composite Web Service Language). Mainly, CWSL is used to describe

- Workflow Structure (transition, sequential, and, or, xor, decision, iteration, start/end blocks, web service template)
- Composition Helper Components (Process Variables)
- Constraints

Exporting language of CWSL is XML (Extensible Markup Language). XML provides a text-based means to describe and apply a tree-based structure to information [50]. Based on the model, a graphical tool (CWSL Designer) is developed for user interaction to define workflow structure, constraints and

references to ontology definition files. CWSL Designer is described in detail in section 4.

3.4.1 Workflow Structure

Following the process definition standards defined by Workflow Management Coalition (WfMC) in [20], workflow specifications consists of basic block structures. In general, a block can have sub blocks; therefore a block structure is a tree which can have recursive structures. Start Block, End Block, Condition and Decision Block can not have sub blocks; these blocks are the execution units that direct the flow of composition and are not included in the scheduling of composition. Similarly, web service template block can not be divided into sub-blocks that models web service template as the simplest block of CWSL.

A root block named as “Composite Process” that extends Sequential block is defined to capture all defined sub-blocks. That is, when a service template block is created, it is automatically placed in “Composite Process” block. In the following subsections, elements of CWSL workflow structure are defined.

3.4.2 Process Variables

Process variables are key-value pairs that maintain information related to the CWSL process. These variables are used to store process data such as evaluation of conditions or reference data flow values such as output of web services, which are passed between web services.

The variable names are referred as String. CWSF Engine supports mainly following java object types;

- java.lang.String
- java.lang.Boolean
- java.lang.Double
- java.lang.Long

- java.lang.Integer
- java.util.Date

By default, any object that implements java.io.Serializable interface can be accessible via CWSF Process Variable Context. In CWSL definition, process variables are defined as follows;

<variable name=”inputX” type=”String” value=”3”/> where;

- *Name* identifies the variable, mandatory field
- *Type* identifies type of the variable, a java type defined above, optionally field
- *Value* is the initializing value of the variable, optionally field

Since many variables can be defined, “process-variable-set” construct is used to encapsulate variables.

```
<process-variable-set>
        <variable name=”inputX” type=”String” value=”3”/>
        <variable name=”outputX” type=”String”/>
        ….
</process-variable-set>
```

In a CWSL file, only string, integer, long, double and Boolean values are allowed. Mainly process variables store information about user such as credit-card number, user international id number, or values used in the composition such as total cost of travel. These values can be used especially when an external interface is integrated with the CWSF Engine. As an example, web interface can be used to take value of process variables from external user.

3.4.3 General Block Structure

Every block structure has some common elements such as transition.

3.4.3.1 Transition

As in Figure 3.3, transition connects “Block A” to “Block B”. After execution of Block A is completed, execution starts with Block B via transition.

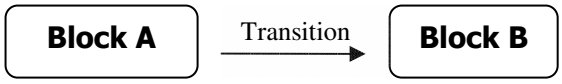

Figure 3.3 Transition

A transition can have a condition element if it is placed in the decision block. If transition has a condition element, the CWSF engine executes the condition expression during pre-scheduling to minimize the number of service templates. If expression is evaluated as true, that transition by the CWSF engine will be executed, and execution will continue with that path. If not, other available transitions are tried to find a successful transition. In the case that every transition condition fails, engine stops the execution in the path. This means that subsequent blocks are not executed. If any of these blocks have a constraint that requires the execution of that block, engine returns no solution to user.

In general transitions are required in sequential and decision blocks. In sequential blocks, sequence of blocks are defined via transitions, and in decision blocks, engine follows the transition that evaluates as true. In Xor, Or, And blocks, transition does not apply any logic, only informs the user.

Below, there is an example of transition where a decision block can start flow of the other blocks via defined transitions.

When a decision block has multiple transitions, transitions are defined in the "transition-set" element.

```
<transition-set>
       <transition name="toflight" to="flight">
         if(choice.equals("flight"))
               return true;
         else
               return false;
       </transition>
       <transition name="totrain" to="train">
         if(choice.equals("train"))
               return true;
         else
               return false;
       </transiton>
       <transition name="tobus" to="bus"/>
</transition-set>
```

In transition expressions, process variables can be used to decide selected transition, as in the above example, "choice" is a process variable.

3.4.3.2 Script Language

As a scripting language, BSF (Bean Scripting Framework) [53] is used to evaluate expressions and change the state of a workflow. BSF is a set of Java classes which provides scripting language support within Java applications, and access to Java objects and methods from scripting languages. Currently Java [54] languages are tested and integrated with the framework; other languages such as JavaScript [55], Pyhton [56], Groovy [57] can be integrated. Default language of expression is Java, and choice of language can be defined as an attribute of expression. In the expression, process variables are used for evaluation. By default, all process variables are included in the expression. In the following example, a simple expression is defined.

```
<expression type="java">
   if(cardType.equals("visa"))
   {
         creditCard = 8859-2122-9920-1111;
   }
</expression>
```

3.4.4 Start Block

Composition execution starts with the only start block placed on a Composite Process block, this block should exist only one time in the model, and engine continues the execution with a transition to the next block.

In the following example, the CWSL structure of the start block is given;

```
<start name="Start">
    <transition name="tohotel" to="hotel"/>
</start>
```

3.4.5 End Block

Composition execution stops when it encounters End Block. Since there can be multiple execution paths, End block can only prevent the execution on the path of the End block, if other parallel paths of the execution exist, they continue normally. In the End block, no transition can be defined.

In the following example, CWSL structure of the "End block" is given;

```
<end name="End">
</end>
```

3.4.6 Sequential Block

In a Sequential Block sub-blocks are executed sequentially in the order of definition. In Figure 3.4, sequential block model is illustrated;

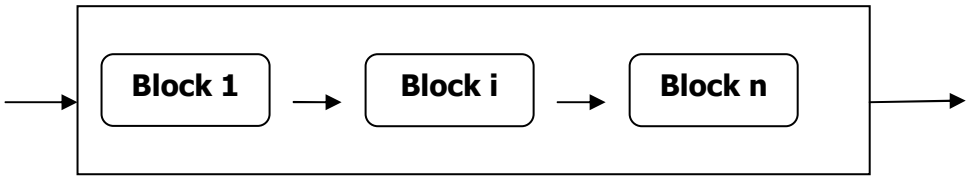

Figure 3.4 Sequential Block

A transition element is not mandatory among blocks in sequential block, but should be used for clarity. However, if transition definitions exist in the sub-block, CWSF engine uses transitions for finding the next block. Otherwise, next defined block is used for execution. In the sequence block, all types of blocks such as “And block”, “Or block”, “Iterate block” can be a block.

In the following example, the CWSL structure of the “Sequence block” is given;

```
<sequence>
        <sequence> …. </sequence>
        <and> … </and>
        <iterate> … </iterate>
        <sequence> … </sequence>
</sequence>
```

3.4.7 And Block

In an AND-block, all sub-blocks should be executed and the block successfully finishes execution when all its sub-blocks complete successfully. Depending on the resources and the constraints, some or all of the sub-blocks can execute concurrently. In Figure 3.5, “And block” model is illustrated;

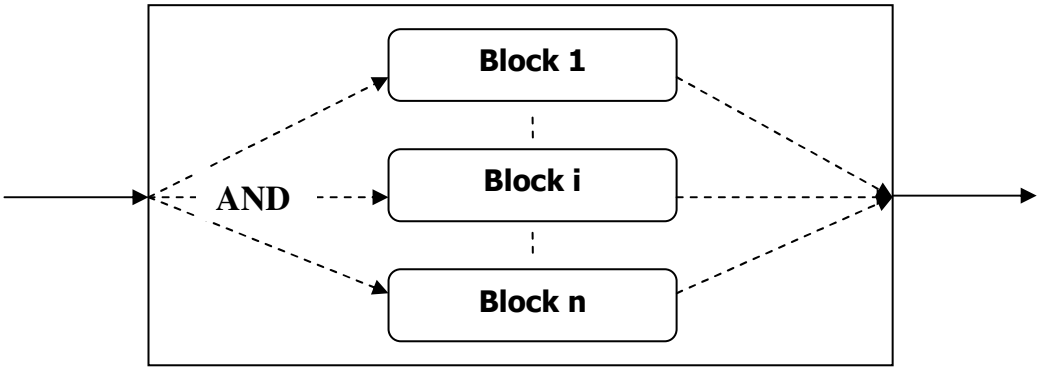

Figure 3.5 And Block

In the following example, a CWSL structure of the "And block" is given;

```
<and>
        <sequence> …. </sequence>
        <xor> …. </xor>
        <or> …. </or>
</and>
```

3.4.8 Or Block

OR-block is completed successfully when at least one of the sub-blocks successfully can complete the execution. In Figure 3.6, the "Or block" is illustrated;

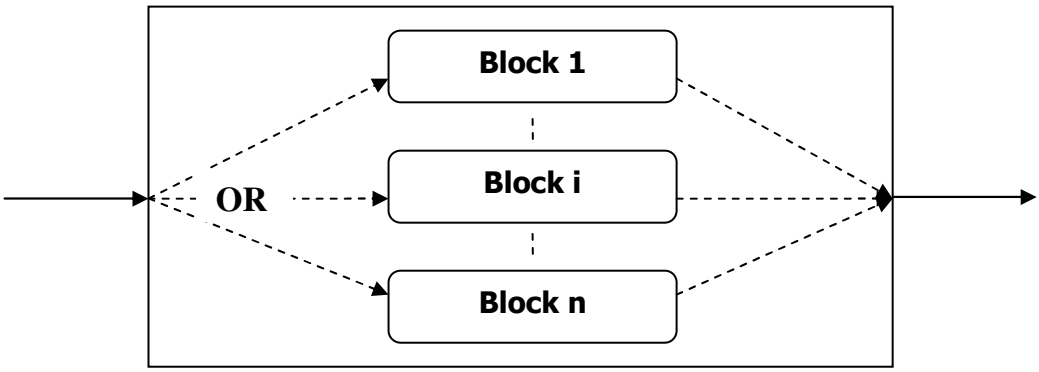

Figure 3.6 Or Block

As an example, person can make some free activities such as ski or balloon in the travel example. In this case, only train, only ski activity can be selected, or both activities can be selected. In the following example, CWSL structure of the or block in this example is given;

```
<or>
        <service-template name=”ski”> …. </service-template>
        <service-template name=”plane”> …. </service-template>
</or>
```

3.4.9 Xor Block

In an XOR-block (eXclusive Or) only one of the sub-blocks can be completed. The block successfully finishes execution when at most and at least one of the sub-blocks completes successfully. “Xor block” is illustrated in Figure 3.7;

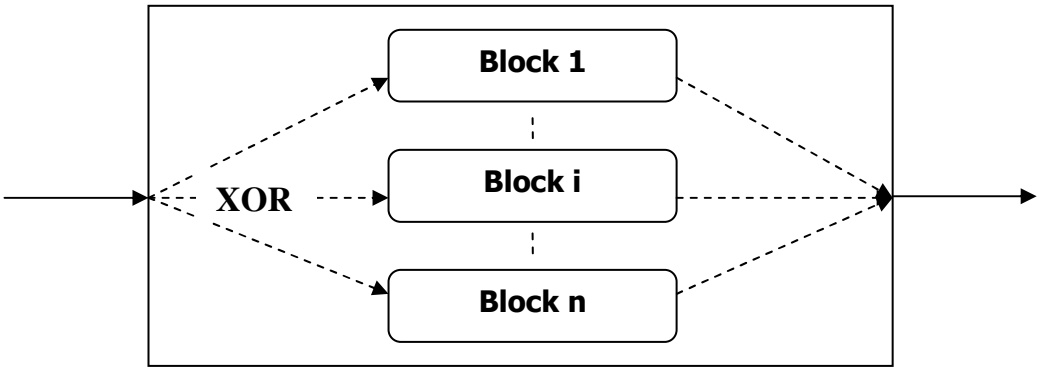

Figure 3.7 Xor Block

In Xor block, all sub-blocks that can have sub-blocks or service-templates can be used. As an example, person in the traveler need to use only transportation method, he can use either train or plane, but he should use one of them. In the following example, CWSL structure of the “Xor block” in Travel Planner example is given;

```
<xor>
        <service-template name=”plane”> …. </service-template >
        <service-template name=”train”> …. </ service-template>
</xor>
```

3.4.10 Iteration Block

Iteration block defines a service template that may run more than once repetitively based on a condition. Thus, several instances of the service template block may execute sequentially. In the scheduling of iteration block, goal is to find the number of iterations of the service template block. Goal can be defined as a constraint that assigns a value to the iteration variable. Iteration variable defines the how many times the block executed by external engine. In Figure 3.8, iteration block model is illustrated;

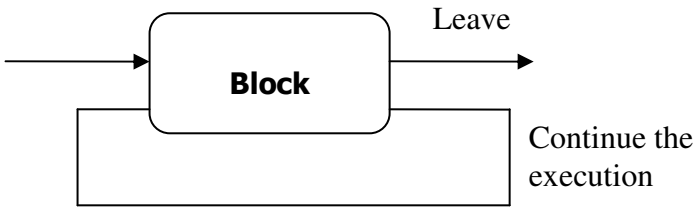

Figure 3.8 Iteration Block

In Iteration block, only one “Service Template Block” is allowed to execute. In the following example, CWSL structure of the iterate block is given;

```
<iterate name=”iteration” assign-method=”minimize” variable=”iteration.count”>
        <service-template>
                ….
        </service-template>
</iterate>
```

Attribute “variable” defines the iteration variable, this variable can be accessed by constraint engine and constraints defined to select values for the variable. “assign-method” is used to control the selected value. If “minimize” used, variable is tried to assign to smallest value of domain that variable can take. If “maximize” used, variable is tried to assign to biggest value of domain that variable can take.

3.4.11 Decision Block

A decision block decides the path of execution in the planning phase to optimize the queries and constraint problem. A decision block continues the execution by using selected transition. Other paths are discarded by CWSF Engine, and service templates in these blocks are not sent any query and variables associated with these templates are discarded in the expressions. First transition evaluated as true selected by engine, if no transitions selected, default transition used by the engine, if no default transition exist, first transition defined selected. In Figure 3.9, decision block model is illustrated;

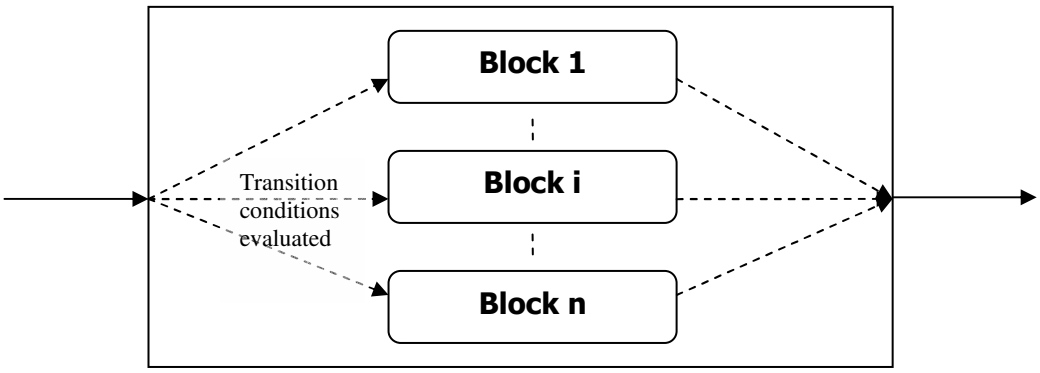

Figure 3.9 Decision Block

In the following example, CWSL structure of the decision block is given;

```
<decision name="decideactivity">
      <transition name="to balloon" to="balloonactivity">
            <expression>
                  return false;
            </expression>
```

```
            </transition>
            <transition name="to ski" to="skiactivity"
default="true">
                  <expression>
                        return true;
                  </expression>
            </transition>
      </decision>
```

Generally decision blocks used when a user defines a parameter via process variable and based on this parameter value, flow of execution is decided.

3.4.12 Condition Block

Condition block decides whether the execution should continue on the path or not in the planning phase. Concept is similar to decision block except that it has a condition element, in which an expression is evaluated, if the condition is met, the block(s) following the condition can be executed and further queried by CWSF Engine. In this block, sub-blocks are not allowed to be defined; only one transition is allowed. In Figure 3.10, condition block model is illustrated;

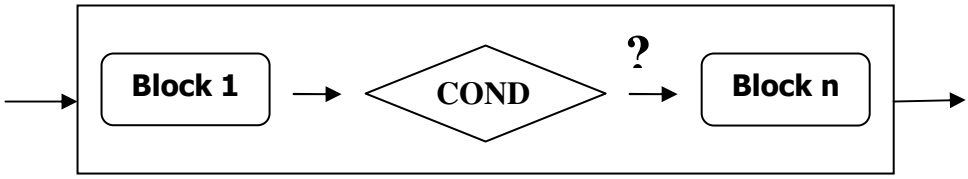

Figure 3.10 Condition Block

In the following example, CWSL structure of the condition block is given;

```
      <condition>
            <expression>
                  if(pathExecuted)
                        return true;
                  else
                        return false;
            <expression>
```

```
            <transition name="tohotel" to="hotel"/>
      </condition>
```

3.4.13 Web Service Template Block

In Figure 3.11, service template model is illustrated;

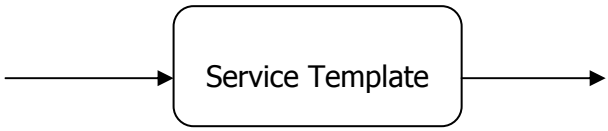

Figure 3.11 Service Template Block

Service Template is an abstraction of expected web service structure. In service template following definitions are available for modeling the expected web service.

- Semantic structure gathered from Semantic Inference Engine
- Expected Query Ontologies from Context Ontology
- Query Constraints for querying and selection of web service
- Constraint and Variable definitions

Below, there is sample structure that show simple service template structure;

```
<service-template name="Hotel">
     <ontology-class>
             <owlClassModel>reshotel</owlClassModel>
     </ontology-class>
     <variable-set>
         <variable assign-method="minimize" name="cost"/>
     </variable-set>
     <constraint-set>
        <constraint name="some">
                  Hotel.cost < 200
        </constraint>
     </constraint-set>
     <query>
        <criteria>
           <queryconstraint attribute="location" op="equal"
                            value="Ankara"/>
        </criteria>
     </query>
</service-template>
```

Details of each structure will be defined in the following sections.

3.5 Constraints

Constraint is a condition that forces web service selection and which must be met during planning process. CWSL supports many types of constraints;

- Temporal and causality constraints
- Resource Allocation Constraints
    - Cost Constraints
    - Logical Constraints
    - If/Then Constraints
    - Control Constraints
- Business Rules

On the contrary to previous web service composition works that deal with temporal/causality constraints, we specify resource allocation constraints as well as temporal/causality constraints and find schedules satisfying all of the given constraints. In addition, business rules are defined to change the state if some condition exists. Before going into the types of CWSL constraint definition library, it is necessary to explain building blocks of constraints.

3.5.1 Building Blocks of Constraint Definition Library

A constraint is simply a logical relation among several unknowns (or variables), each taking a value in a given domain. A constraint thus restricts the possible values that variables can take; it represents some partial information about the variables of interest. To represent web service composition problem as a constraint satisfaction problem; it is necessary to create variables with a domain, which will be constructed from candidate web service values. In the following sub-sections, how CWSL defines and constructs variables for composing web services.

3.5.1.1 Variables

Variables can be integer or string based variables. In CWSL, multiple variables are defined in structure “variable-set” in the service template. In the following example, cost is defined as an integer based variable, and location defined as a string variable.

```
<variable-set>

     <variable assign-method="minimize" name="cost" type="Int"/>

     <variable assign-method="minimize" name="location"
type="String"/>

</variable-set>
```

Generally, variable domain constructed as follows. Each web service template defines the action ontology which defines expected query ontology of candidate web services. Candidate web service that does not provide the requirements is discarded by Semantic Matching/Mapping Engine. Information gathered by candidates is stored by CWSF Engine Data Storage Module. This modules under the request of a variable, provides domain that generated by iterating over every candidate value. For integer variables, value from candidate is gathered directly. On the other hand, if it is string based variable, it is required to convert string value to integer value because constraint engine can handle integer expressions naturally. Therefore, CWSF engine assigns each string value a unique integer value. Then, string variable will have a domain that has integer values which represents string values from candidate web services. For example, for location attribute, “Antalya” value is converted to value “17” in CWSF Engine. Engine assures that assignment of each value is unique. Therefore, there will be no wrong implication by both constraint engine and CWSF Engine.

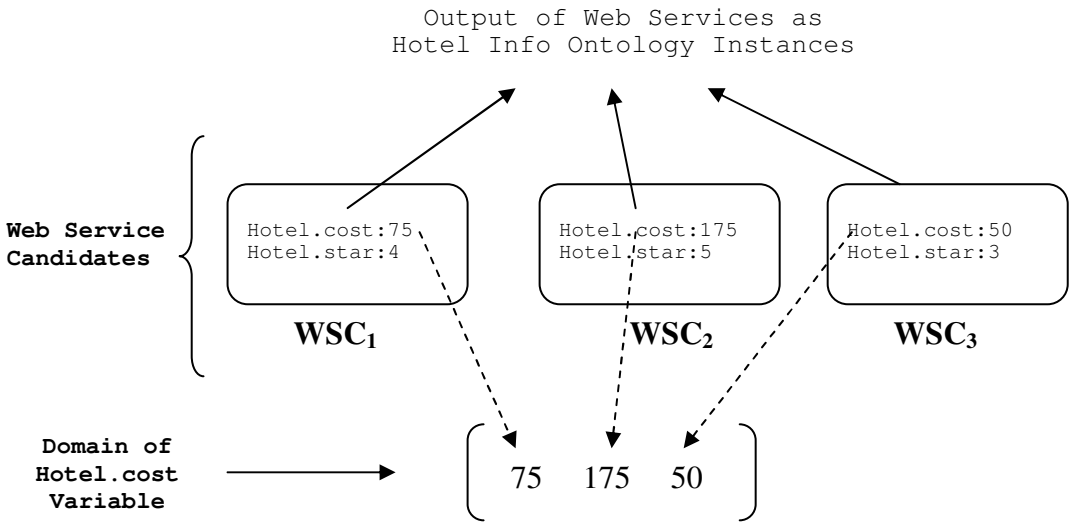

Figure 3.12 Variable Creation

Variable creation of Hotel Reservation web service template is shown in Figure 3.12. By Semantic Inference Engine, action ontology of web service is defined as "Hotel Reservation". In action ontology, the query ontology of service template described as "Hotel Info" ontology. Therefore, semantic matching/mapping engine ensures that every candidate satisfies the required ontology structure for querying. Therefore, every web service candidate returns values in the same ontology class. In the example in Figure 3.12, WSC_1, WSC_2, WSC_3 candidate's produces 75, 175, 50 values for the cost attribute of "Hotel Info" ontology. These values form the domain for selection of a value by the constraint engine. This domain used by variable as a possible set of values. In the solution, CWSF engine use assigned variable value for selection of web services, or gathering information to constraint engine solution to web service composition solution.

3.5.1.2 Operations

Operations are used to form complex constraints or expressions to make more powerful language. Operations are divided into three; Arithmetic, Boolean and Comparison Operations.

3.5.1.2.1 Arithmetic Operations

To make expressions with variables and constraints, it is necessary to use arithmetic operations. All operations are binary, and following arithmetic operators are supported;

- + | - | *

A simple arithmetic example in an expression is;

(Hotel.cost – (Transportation.cost + Ski.cost*2))

3.5.1.2.2 Comparison Operations

Comparison operators are necessary to create simple constraints between expression, variables, and constant values.

- < | > | <= | >= | == | !=

The simplest constraint is a comparison between two atomic expressions. More complex constraints can be specified by combining simpler constraints. A simple example with using comparison operators is follows;

(Hotel.cost + Ski.Cost) < Transportation.cost

3.5.1.2.3 Boolean Operations

Boolean operations like “and” are used especially define more complex logical expressions. “and”, “or”, “xor” are binary operations, however “not” is unary operation that accepts one operand. Following boolean operators are supported;

- and | or | xor | not

A simple logical constraint is;

```
(Balloon.cost < 100 and Ski.cost < 75) or
(Hotel.cost + Ski.Cost + Transportation.Cost > 600) and
((not)(user.money > 500))
```

3.5.1.3 Expressions

In CWSL, same expression can be used by many constraints. This provides flexibility for using expressions in different constraints, in business rules, also as a sub expression in different expressions. Expression are referred by "$" sign, CWSL engine when discover by $ sign, place that expression in the current expression, constraint or business rule. In the following example, expression referred by a constraint.

```
   <expression name="totalExp">
      Hotel.cost + Ski.cost + Balloon.cost + Train.cost + 
Flight.cost
   </expression>
   <constraint>
      $totalExp < 1000
   </constraint>
```

3.5.2 Temporal and causality constraints

As explained in section 3.4, business logic of composition workflow is generally specified by using block structures. Generally, these types of web service composition approaches [6, 25] capture most of the order and existence dependencies among web services. However, there may be other order and existence dependencies that can not be defined using these structures, such as the dependencies between two web services in different service template blocks. The typical examples of such temporal/causality constraints may be listed as follows:

- Web service w_1 must execute.
    - CWSL representation: <execute service="Hotel"/>

- Web service w_2 must not execute.
    - CWSL representation: <notexecute service="Hotel"/>

- If web service w_1 executes, then web service w_2 must execute as well. (Klein's existence constraint in [61])
    - CWSL representation: <ifexecute service="Hotel" then="Ski"/>

3.5.3 Resource Allocation Constraints

On the contrary to general web service composition methods, CWSL specify resource allocation constraints to find schedules satisfying business constraints defined by user. Resource allocation constraints define restrictions that how web service candidates selected for web service template. A resource allocation constraint may be any factor and rules that affects the selection of a web service for a given process.

The simplest constraint is a comparison between two atomic expressions. More complex constraints can be specified by complex expressions and combining simpler constraints.

Some of the constraints directly involve the total execution cost, whereas others define restrictions on web services and relations between web services. From this point of view, we may categorize the resource allocation constraints as cost and control constraints.

3.5.3.1 Cost Constraints

In Travel Planner example, in this example user will accommodate in a hotel, he travels either by train or by plane, and he can make free activities such as balloon, or ski, or both. User's constraint define that total cost of all travel should not cost more than $1500. Therefore, a typical example of a cost constraint is the following expression in CWSL language;

```
<cost-constraint>
        Hotel.cost + Plane.cost + Train.cost + Balloon.cost + Ski.cost < 1500
</cost-constraint>
```

Since only one transportation method can be selected, either Plane.cost or Train.cost will be forced to value of zero by the CWSF engine.

3.5.3.2 Logical Constraints

These constraints are any logical expressions on integer or string values. Any expression with any depth can be valid for CWSF system. As a simple example, user can define a constraint that requires 5 star Hotels with cost less than 250$ per daily or 4 star hotel with daily cost less than 150$ and located in town. This example can be modeled by the following expression.

<constraint>
(Hotel.star == 5) || (Hotel.dailyCost < 250 && Hotel.location == “Ankara”)
</constraint>

As in the following example, “Hotel.location” is a string variable and “Ankara” value is a string expression, Hotel.location == “Ankara” statement requires location value equals to “Ankara”. Therefore CWSF engine construct integer domain for “Hotel.location” and assigns “Ankara” to integer value, therefore this expression converted to integer expression. This is explained in detail in section 3.5.1.1.

3.5.3.3 If/Then Constraints

These type constraint only apply when the constraint clause in the “if” part satisfied, constraint clause in the “then” part is forced to be satisfied. Structure of the if/then constraint is;

If C_X *Then* C_Y where C_X and C_Y are hard resource constraints.

A simple example of constraint is;

If Ski.cost > 250 Then Balloon.cost < 300

CWSL representation of this constraint is;

```
<ifthen>
        <if>Ski.cost > 250</if>
        <then> Balloon.cost < 300</then>
</ifthen>
```

3.5.3.4 Control Constraints

Control Constraints are defined directly on web service templates. If web service template WT_1 is done by web service W_1, then web service template WT_2 must be done by the same web service as well is an example for control constraints. This control constraint can be explained in CWSL as follows;

```
<control type=”same” template1=”wt1” template2=”wt2”/>
```

In this definition, template names are identified template1 and template2 fields. Type attribute identifies category of control constraint, it can be either “same”, or “different”. If same attribute is placed, then CWSF engine force constraint engine to select the same web service provider for defined service templates, otherwise different providers are selected for each service template. Web service provider definition is gathered from UDDI Service registry and it is stored as a string. Therefore CWSF Engine has Service Provider Registry that stores information about service providers and assigns them unique identifier.

3.5.4 Business Rules

They are the rules that facilitate the constraint given in the antecedent of the rule, if the precedent condition is fulfilled. Business rules are different from the constraints that business rules are valid if the specified condition in the solution exists. On the other hand, constraints strictly define rules that should evaluate as true. An example is as follows:

- If both ski and balloon activities exist in holiday program, then total cost can cost $100 more.

In CWSL, business rule are defined on an expression, when “if” construct evaluated as true, business rule would be true, and “then” part is executed. Following code snippets show expression that business rule apply, ant the rule itself.

```
<expression name=”totalExp”>
    Hotel.cost + Ski.cost + Balloon.cost + Train.cost + Flight.cost
</expression>

<business-rules>
  <businessrule name="freeActivityRule" onexpression="totalExp">
    <if>ski.executionStatus == 1 and balloon.executionStatus ==
1</if>
    <then>$totalExp + 100</then>
  </businessrule>
</business-rules>
```

CWSL Business Rule library support summation, subtraction, and multiplication operations on the expressions defined.

3.5.5 Formal Model of Constraints

Formal structure of CWSL Constraint Definition Library Language consist of “Constraint”, “Expression”, “Arithmetic-Operation”, “Comparison-Operation”, “Boolean-Operation”, “Variable”, and “Atomic” elements. Structure of each element is defined below.

```
<Constraint> ::= <Constraint><Boolean-Op><Constraint> |
                 <Expression><Comp-Op><Expression>
<Expression> ::= <Expression><Ar-Op><Expression> |
                 <Variable>|<Atomic>
<Ar-Op>      ::=  + | -
<Comp-Op>    ::= < | > | <= | >= | == | !=
<Boolean-Op> ::= and | or | xor | not
<Variable>   ::= <a set of possible values including integer or
            string
                  values>
<Atomic>     ::= integer number, string value
```

This formal model can be represented as a tree expression. In Figure 3.13, following simple constraint is parsed and processed by CWSF engine. Engine traverses tree

from deepest level to until reach to root. In other words, all nodes in the tree at depth n are parsed, and then all nodes at depth n -1, this continues until root. During traversal, variables and atomic values are inserted into temporary expressions, sub-constraints are created, and finally one constraint defines user's all constraint are inserted to constraint engine.

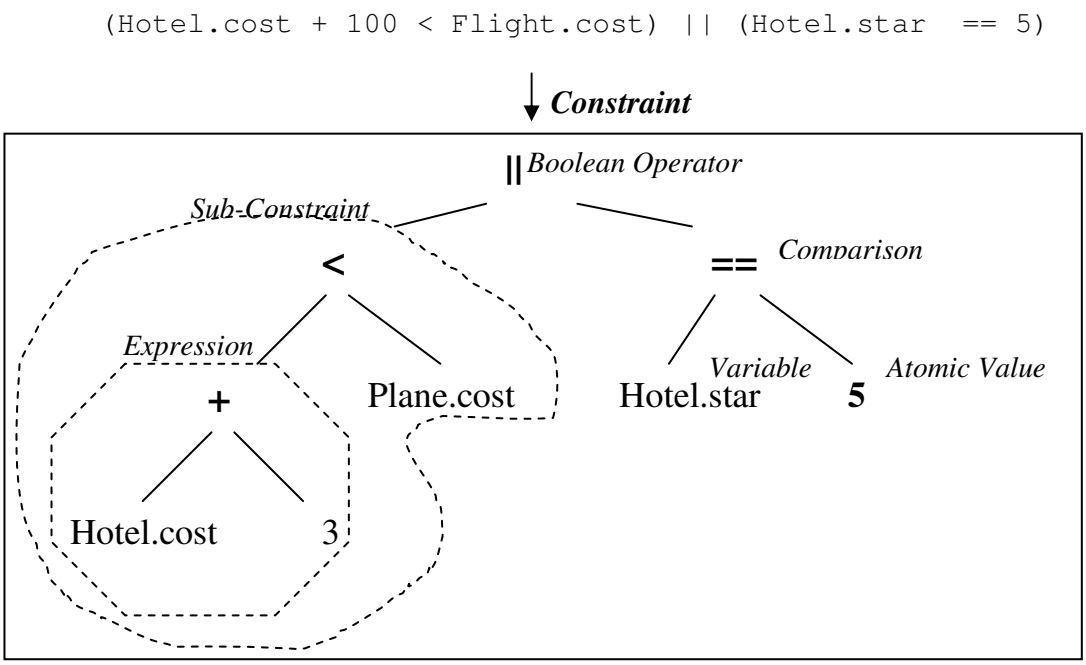

Figure 3.13 Constraint Creation

3.6 Semantic Inference Engine

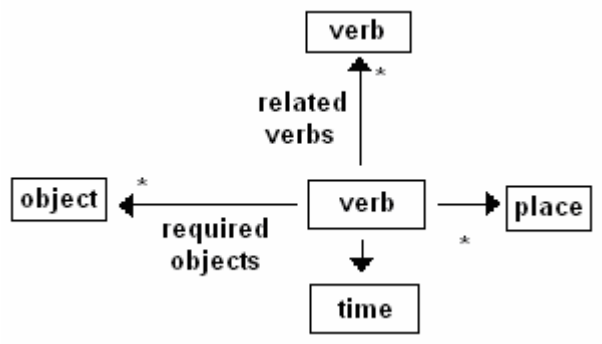

Figure 3.14 Context Model

To compose semantic relations between concepts in different ontologies, context model constructed which defines relations between ontologies.

In this model, actions are taken as the core of the context. Therefore, the contexts are defined around verbs and the relations among verbs. Verb is the part of speech that expresses existence, action, or occurrence in most languages. To perform a verb, some objects required. Also these actions are done at certain places. The context model is shown in Figure 3.14. Many-to-many relations defined between verbs in context model. Each context has its related set of verbs. A verb can have different meanings; therefore, it can be related with more than one context. By using the relationships defined in context model, it is possible to infer context and action information. The inferred objects and verbs are associated with ontology definitions and this information is used for guiding the user in modeling the composite process. This part is not in the scope of this thesis, and Kardas [58] explains in detail how matching/mapping capabilities can be used to achieve composition.

3.7 Service Discovery

Although there are efforts to incorporate semantic information with service registries as in [11], current UDDI registries can only do keyword based search. There is not yet any mechanism to define relationships among web services since there is not any meta-data about services in the registries. CWSF system also use keyword based search in UDDI registry. However, to add semantic capabilities, CWSF add Semantic Wrapper Layer which is an adapter of this registry. CWSF assumes that service's semantic descriptions (OWL-S file) hold in UDDI registry. By this method, after a keyword search, semantic descriptions of services can be accessed. In the Figure 3.15, how CWSF engine interacts with UDDI server is illustrated.

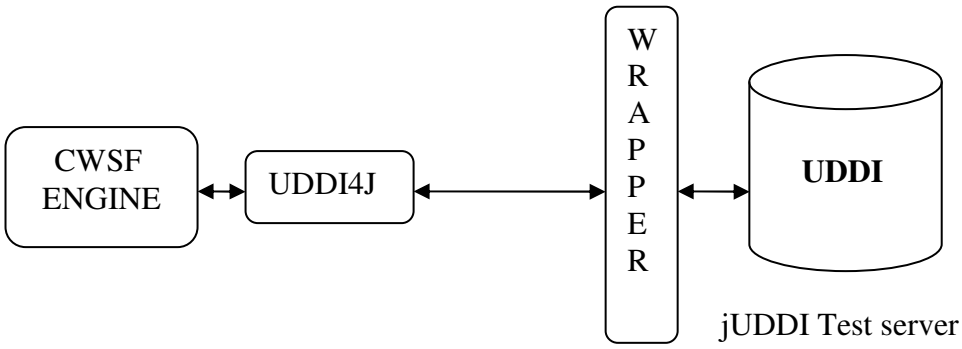

Figure 3.15 UDDI Search

On querying registries, uddi4j library used, which is a Java library that provides an API to interact with a UDDI registry [51]. On simulating UDDI server, jUDDI [52] test server is set up, that provides open source Java implementation of the UDDI specification for Web Services. jUDDI is a pure Java web application and as such can be deployed to any application server or servlet engine that supports version 2.1 or later of the servlet API.

3.8 Matching/Mapping Engine

Matching/Mapping Engine of CWSF sub-system matches these services and finds their interoperability by using semantic matcher module. CWSF discover services

with complementary functionalities. CWSF compare the syntactic and semantic features of web services to determine whether two services are composable. We do this semantic matching (capability matching) and determine provided capabilities according to the given needed capabilities.

Initial semantic matching algorithm was based on the algorithm given in [10, 12], in which concepts are compared according to their hierarchy in ontology model. Approach extended with the context model to be able to infer further relationships among ontologies.

Semantic mapping module provides mappings between the same/similar concepts in different ontologies. Same ontologies can be defined in different languages. In this case with the help of the dictionary services the equality of the ontologies can be decided. In this case synonyms are taken into consideration as well. CWSF not only compare concept names but also their attributes' names and types.

After possible solutions gathered from constraint engine, matching/mapping engine works for input/output matching to find executable web service composition. In the case that no input defined for execution service, then this input can be requested from the user. For example, user credit card number can be requested from user. This part is not in the scope of this thesis, Kardas [58] explains in detail how matching/mapping capabilities can be used to achieve composition.

3.9 Service Query Engine

Service query engine is responsible for gathering information from candidate web services. Service queries are generated based on the semantic attributes in CWSL. The results of the web services are stored and processed by the plan generator. CWSF standardize the query structure of the web service. Assumption is based on the structure of the input and output of the web service. Each web service takes "QueryCriteria" object and returns the semantic concept defined in action ontology such as "Hotel Booking Info" that web service provides. Semantic concept of web

service is determined by semantic inference engine. In Figure 3.16, query service of web service is illustrated;

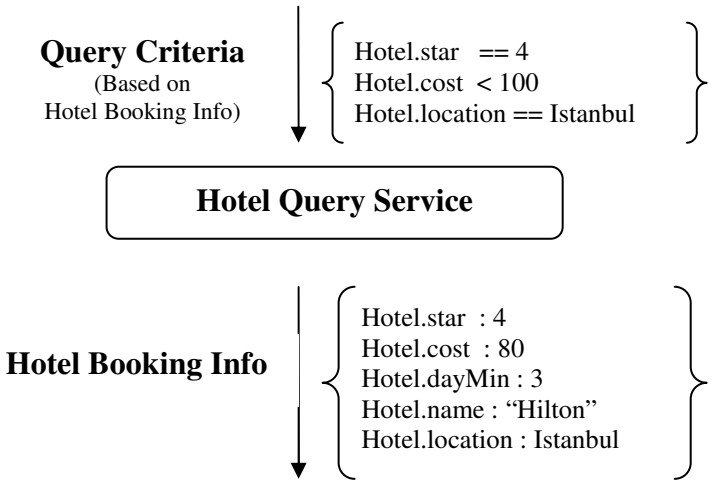

Figure 3.16 Query Example

QueryCriteria structure consist of "QueryConstraint" objects, each of them defines a constraint on the selection of a web service. QueryConstraint object structure has the following building elements;

- Name: Referred attribute of semantic structure,
- Operation: the rule that how query service interpret the constraint, excepted operations are <, >, <=, >=, !=, ==
- Value: any string, integer value that will be compared against the attribute of semantic structure.

Mainly, Query Constraints are used to help service candidates to return more meaningful answers. It is assumed that understanding query constraints is the responsibility of the service candidates.

After query web service invoked with QueryCriteria structure, output of web service query, OWL Instance is stored in Data Storage Module for further analyzing and operations such as creation of variables and constraints in constraint engine.

Also, to assure validity of candidate web services, all query constraints are evaluated for output of Service Candidate. If it does not satisfy the requirements, service provider is removed from candidate set.

3.10 Plan Generator

Plan generator module creates plans using constraint optimizer. Constraint translator in the module converts CWSL workflow structure, web service query outputs and constraints to the language of the constraint optimizer. Output of the constraint optimizer is evaluated by plan generator module. The constraint optimizer makes sure that the discovered services satisfy the client request and then optimizes the service sets according to its requirement. Constraint optimizer uses Choco [9] constraint library for the evaluation of constraints. Besides the best solution, it is possible to get all solutions, or the best solution found by the constraint solver based on the defined variable heuristics. Solution analyzer is responsible from translating the answer of constraint engine to answer of service composition problem. After possible solution set is generated by solution analyzer, semantic matcher/mapping module responsible for adjusting the input ontologies of service templates.

3.10.1 Constraint Translator

One of the most important contributions of this work is to view the web service composition problem as a constraint satisfaction problem. Therefore, an important module of the architecture is the constraint translator that converts workflow of web service composition and CWSL constraint specifications into a constraint language of Choco.

In general, any constraint language can be used as target language. Mapping for each workflow block and constraint type specification in CWSL is defined in a constraint structure. By using this mapping, CWSF Engine produces a constraint set equivalent to CWSL specification. In block conversions, if block executed, then conversion rules will be valid, otherwise constraints of these blocks will not be

forced to constraint engine. In the rest of this section, the mapping from CWSL to constraint language is described. Classical mathematical notation for the representation of constraints is used as in defined in section 3.5.5.

3.10.1.1 Consistency of Variables

Constraint engine try to assign valid values to variables, and naturally every service template can have more than two variables. For example, Hotel Booking template can have two variables named as cost and star (indicating cost of hotel and star of hotel). Constraint solution should guarantee that assignment of values should not conflict with each other. That is, selection of values in the cost and hotel variables should point out the same web service candidate. Since constraint engine does not know the concept of web service concept, additional constraints should be inserted to constraint engine for computing valid solutions.

Assume that, there exist three candidate web services with the following cost, star and distance values;

- *Candidate A;* cost:75, star:4, distance:12
- *Candidate B;* cost:80, star:4, distance:7
- *Candidate C;* cost:120, star:5, distance:7

For a solution to be valid, “cost:75” selected from Candidate A and the value in variable “star:4” is selected from Candidate A and “distance:12” is selected from Candidate A, This can be modeled by a “and” structure. Therefore following constraint is necessary to assure the consistency of Candidate A;

cost == 75 *and* star == 4 *and* distance==12

There will be similar constraints for Candidate B and C, and these constraints define a valid execution path. When one of them is true, solution can have a solution. Therefore, “or” logic should be used to assure that constraint engine can find a path to solution. As a result, following constraint will be inserted to engine.

(cost == 75 *and* star == 4 *and* distance==12) *or* (cost == 80 *and* star == 4 *and* distance==7) *or* (cost == 120 *and* star == 5 *and* distance==7)

When there exist only one variable in a service template, it is not necessary to insert such a constraint.

3.10.1.2 Block

In order to capture inter and intra-block dependencies, start, end and duration attributes are defined for blocks, denoting beginning, end and duration of block's execution. Therefore, start and end variables defined and default constraint for each block is;

$$block.start + block.duration = block.end$$

If the execution status of a block is set to zero, this should be reflected to all sub-blocks of this block. Hence, the following constraints are defined for composite blocks (blocks consist of sub-blocks).

$$\forall B, B \text{ is a block, } \forall Q, Q \in B,$$
$$B.executionStatus = 0 \rightarrow Q.executionStatus = 0$$

In the constraint engine, start and end variables are created. To minimize the solution space, it is necessary to assign minimum domain set to these variables. Therefore, for each block, maximum number of blocks that can be executed is taken to calculate the domain.

3.10.1.3 Sequential Block

In sequential block, start time of first block of sequential block should equal to start time of sequential block, and similarly end time of last block should equal to end time of sequential block. Also, for subsequent blocks, execution time of each block should come before next block in the sequence. Sequential block *S* with the sub-blocks $b_{1,}... b_{i,}... b_n$ is defined as:

$S.start = b_1.start$

$S.end = b_n.end$

$\forall$ i, $1 \leq i \leq n-1$, $b_i.end \leq b_{i+1}.start$

3.10.1.4 AND Block

In and block, start time of sub blocks should bigger or equal than start time of and block, and similarly end time of and block should bigger or equal than end time of sub-blocks. And block *A* with the sub-blocks $b_1, \ldots b_i, \ldots b_n$ is defined as:

$\forall\ i, 1 \leq i \leq n, A.start \leq b_i.start$

$\forall\ i, 1 \leq i \leq n, b_i.end \leq A.end$

3.10.1.5 OR Block

OR block *O* with the sub-blocks $b_1, \ldots b_i, \ldots b_n$ is defined as:

$\forall\ i, 1 \leq i \leq n, V\ (O.start \leq b_i.start\ \Lambda\ O.end \geq b_i.end\ \Lambda$ b_i.executionStatus = 1*)*

In an OR block, at least one sub-block is executed. This is modeled through disjunction.

Typically, if a block is executed, its start time is an integer value within the execution time scale. In this representation, a web service is executed if its start time is within the execution time range of the block in which it is defined.

3.10.1.6 XOR Block

XOR block X with the sub-blocks $b_1, \ldots b_i, \ldots b_n$ is defined as:

$\forall\ i, 1 \leq i \leq n,$ V((X.start = bi.start Λ X.end = bi.end) Λ
$\forall_j\ 1 \leq j \leq n, j \neq i$ (bj.executionStatus = 0))

In an XOR block, only one of the sub-blocks is executed. The first part of the constraint shows that a sub-block is chosen for execution and the second part guarantees that once a sub-block is executed, the others are not executed. We represent that a sub-block (or task) is not to be executed by setting block execution

status variable as zero, flagging as not selected. This may be a negative value or a value that is higher than the end of scale. The sub-block is chosen non-deterministically and this is represented by disjunction.

3.10.1.7 Iteration Block

In order to determine the number of iterations, iteration variable assignment should be solved; the value of variable indicates the number of execution of the web service.

3.10.1.8 Temporal and Causality Constraints

In order to represent temporal/causality constraints, block execution status variable is used in order to denote that whether block is executed or not executed. The temporal/causality constraint definitions of CWSL are translated into constraint language as follows:

- *<execute service="Bi"/> is* represented as (B_i*.executionStatus = 1*).
- *<notexecute service="Bi"/>* is represented as (B_i*.executionStatus = 0*).
- <ifexecute service="B_x" then="B_y/> is represented as (B_x*.executionStatus = 1*) → (B_y*.executionStatus = 1*)

3.10.1.8.1 Configuring Cost Constraints

When a web service or block is not selected by constraint engine, related cost constraints should have no wrong effect on the solution. For example selection of Plane Web Service Template implies the not selection of Train Web Service Template. Therefore, variables of Train Web Service Template should not affect the total cost of the execution in Travel Planner example. Value of these variables should equal to 0. For constraint engine, it is not possible to include "0" value for the domain of variable which includes only the values that comes from web service candidates. If zero value is set, then constraint engine can select "0" for each variable to minimize the problem that result in wrong solutions. To eliminate this, a temporary variable approach is used. In this approach another variable is created

with domain of original variable plus zero value. An example of Train.cost variable is given below;

- Domain of Train.cost : *50, 75, 80, 100*
- Domain of temp.Train.cost: ***0****, 50, 75, 80, 100*

Temporary variable with name temp.Train.cost is created and will be replaced in all places of Train.cost where Service Template is placed in an XOR/OR block or block has execution status constraint. Additional constraint is added to satisfy the problem requirements, which forces the multiplication of execution status variable and original variable is equal to temporary variable. When service template is selected for execution, then executionStatus variable will be equal to one, and multiplication result in a value in the domain of original value. Otherwise, executionStatus variable is set to value zero, and expression that contains original value is not effected since temporary value will be zero automatically. Above example is converted to constraint as follows;

Temp.Train.cost = Train.executionStatus * Train.cost

3.10.1.8.2 Configuring Control Constraints

For each web service template, there exist constraint variable with structure "Block Name".executer. Domain of this variable is constructed with the unique ids of candidates provided from Web Service Provider Registry of CWSF Engine.

When control constraint has type "same" as in following example;

<control type="same" template1="Hotel" template2="CarRental"/>

Hotel.executer and CarRental.executer variables are used, and following constraint is posted to engine.

Hotel.executer == CarRental.executer

If constraint is type of different as in following example;

<control type=”same” template1=”Balloon” template2=”Ski”/>

Balloon.executer and Ski.executer variables are used, and following constraint is posted to engine.

Balloon.executer != Ski.executer

3.10.2 Solution Analyzer

Since constraint engine just assign values to variables, it is the responsibility of solution analyzer to understand the output of constraint engine. Therefore, web service composition solution extracted from assigned values. Following information is utilized;

- From variable assignments, mappings are used to detect which web service selected.
- Block execution status values indicates whether blocks are enabled or not, and also sub-blocks including web services.
- Start and end time values determine ordering of web services, therefore scheduling of web services is guaranteed.

It is not necessary that every constraint engine solution is a web service composition solution. There exist controlling variables that can be assigned different values. Since these values can take different values, this increases the number of solutions. Therefore every constraint engine solution can not produce unique solution for CWSF Engine. A Therefore an elimination step is required to analyze all solutions for identifying unique solutions.

When solution is found, selected web service candidates replace service templates in the solution model. After solution model found, service matching/mapping engine works for input/output matching to produce executable schedules.

3.11 Experiments and Efficiency Issues

For most of the constraint problems, the aim is to find the optimal solution which requires the search throughout the whole search space. Therefore, it is necessary to conduct some experiments to check the feasibility of constraint programming approach for web service composition under resource allocation constraints. Experiments focus on three different aspects that affect the execution time for finding a solution for composition;

- *Number of Candidate Web Services*;
    - When the number of candidate web services affects the domain set of variables, and enlarge the search space
- *Model Complexity;*
    - When the Number of blocks increases in the model, the number of variables and constraint in problem also increases.
- *Variable/Constraint Number*
    - Increasing the number of variable and/or constraints directly affects the solution finding.

Since it is not possible to simulate many web services manually, a simulator is constructed for experiments. This system simulates randomly the outputs of web services selection process for given service template definition. All experiments are conducted on "AMD Turion 64 x2 TL50 Mobile (1.6 ghz)" processor with 1 GB RAM running on Windows XP. Execution time for solution includes conversion of web service composition into constraint satisfaction problem, constraint engine execution to find solution. In all experiments, execution times are taken the average of running experiment setup for ten times.

3.11.1 Experiment: Number of Candidate Web Services

First experiment type is conducted to observe how increasing number of candidate web services affect the solution time under a simple CWSL model. CWSL model consist of a sequence block that includes five service template blocks and five constraints. In Figure 3.17, CWSL model that is used in the experiment is illustrated.

Figure 3.17 CWSL Model for Candidate Web Services Experiment

Experiment performed against candidate ten, hundred, thousand number of candidates conducted respectively. In the following graph in Figure 3.18, execution time versus number of candidate web services is given as a line.

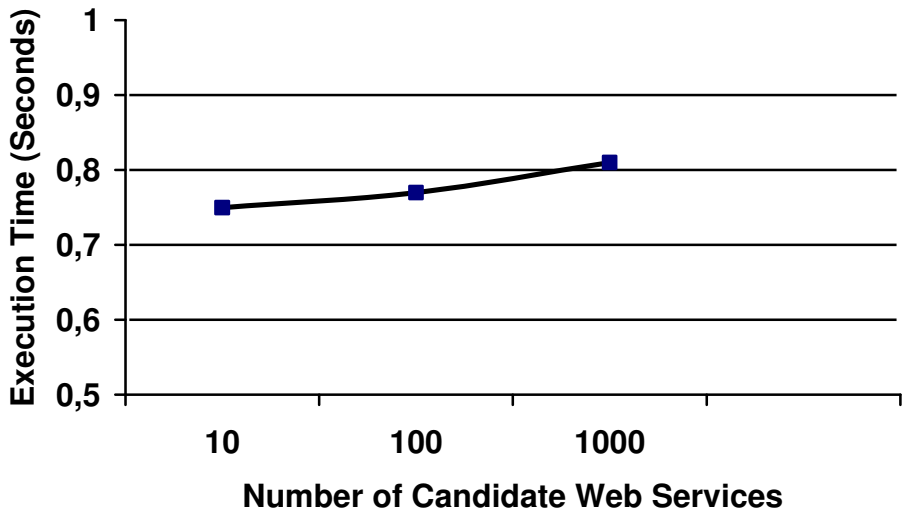

Figure 3.18 Execution time versus Number of Candidate Web Services

As graph shows, execution time is slightly increased when we increase the number of candidate web service. When candidate web service number is ten, execution time is 0,75 seconds, and when candidate set is about one thousand, execution time is 0,81 seconds. This shows that when we increase by 100 times, execution time increase only 0,06 seconds. Therefore, the number of candidate web service has a little effect on the solution, and CWSF model can use many candidate web services without facing important performance problem.

3.11.2 Experiment: Model Complexity (Number of Blocks)

Second experiment display when model complexity increases with the number of block hierarchies, that is how changes the execution time when we increase the total number of "and", "or", "xor", "sequence", "iterate" and "service template" blocks. In this experiment, total number of blocks changes from twenty, fourth and sixty, respectively.

In this experiment, no constraint added to model and for every service template there exist three variables and fifty candidate web services exist. Every twenty block has structure in Figure 3.19.

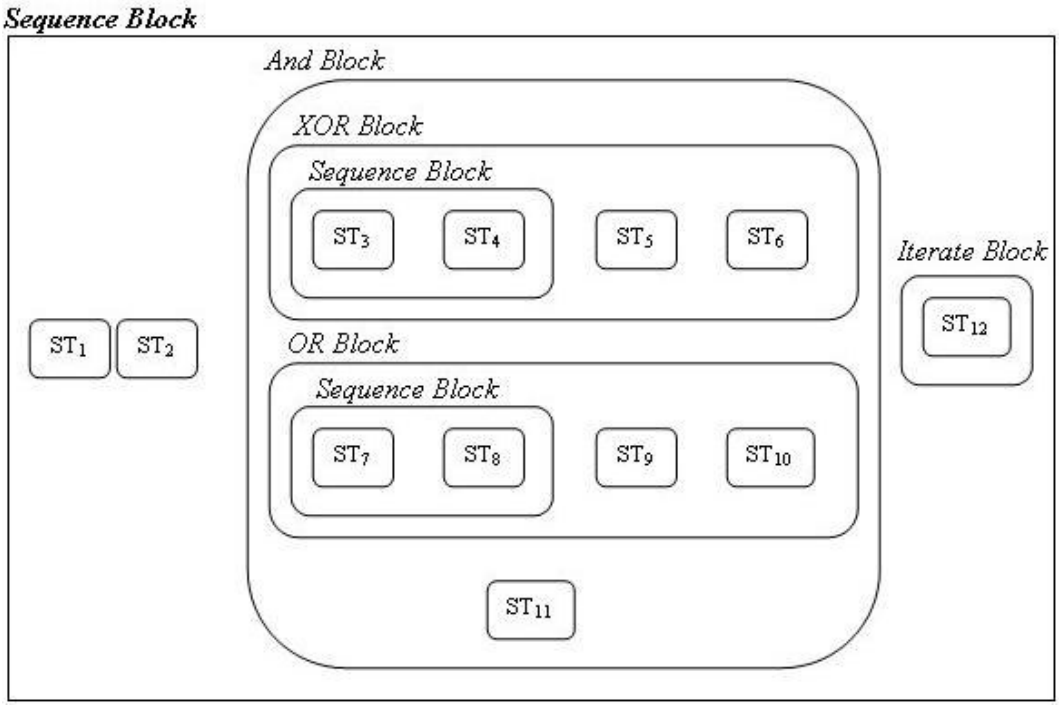

Figure 3.19 CWSL Model for Model Complexity

In Figure 3.20, there is a graph that displays execution times versus by the number of blocks;

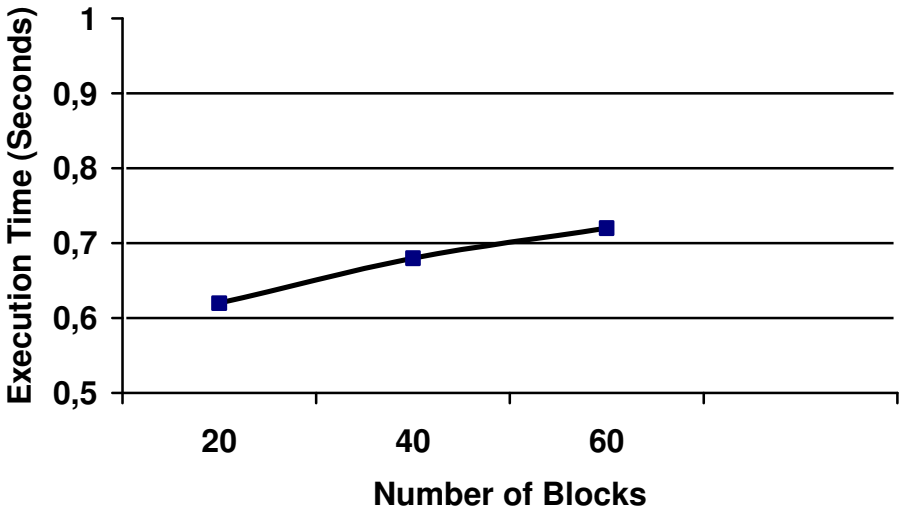

Figure 3.20 Execution time versus Number of Blocks

As seen in the Figure 3.20, execution time increases linearly with the number of blocks. For twenty blocks, time is 0.62 seconds, for fourthy blocks, execution time is 0.68 seconds, for sixty blocks execution time is 0.72 seconds. Therefore increasing the number of blocks does not affect the performance.

3.11.3 Experiment: Number of Variable/Constraint

The number of constraints or variables directly affects the performance of both CWSF Engine and constraint engine. An experiment conducted on the CWSL structure defined in Figure 3.20. During experiment, each web service template has fifty candidate web services, and for each web service template, there can be at most four variables. In each try, variable on service template increased by one. Generally, constraints has inside expressions like "((st7.cost + st8.cost) > 100) || ((st4.cost > 10) && (st5.cost > 10))" where "cost" are variables of service templates, these expressions are generally complex. In this experiment, variable number by Service Template (ST), total numbers of Service Template (ST) variables and user constraints manually added to system in addition to experiment are follows;

- Try 1; ST Variable:1, Total ST Variable: 12, User Constraint: 10
- Try 2; ST Variable:2, Total ST Variable: 24, User Constraint: 20
- Try 3; ST Variable:3, Total ST Variable: 36, User Constraint: 30
- Try 4; ST Variable:4, Total ST Variable: 48, User Constraint: 40

For each variable set (includes twelve variables), ten constraints over these variables added to problem. In addition to these constraints, CWSF engine, add controlling variables and constraints to ensure consistency of variables, and scheduling the web services. Except first try which has only one variable in service template, fifty constraints are added to engine for ensuring that constraint engine should not conflict with the web service composition problem. In each try, constraints are become more complex when we increase variable number.

In Figure 3.21, graph of execution time versus number of variable/constraint are displayed.

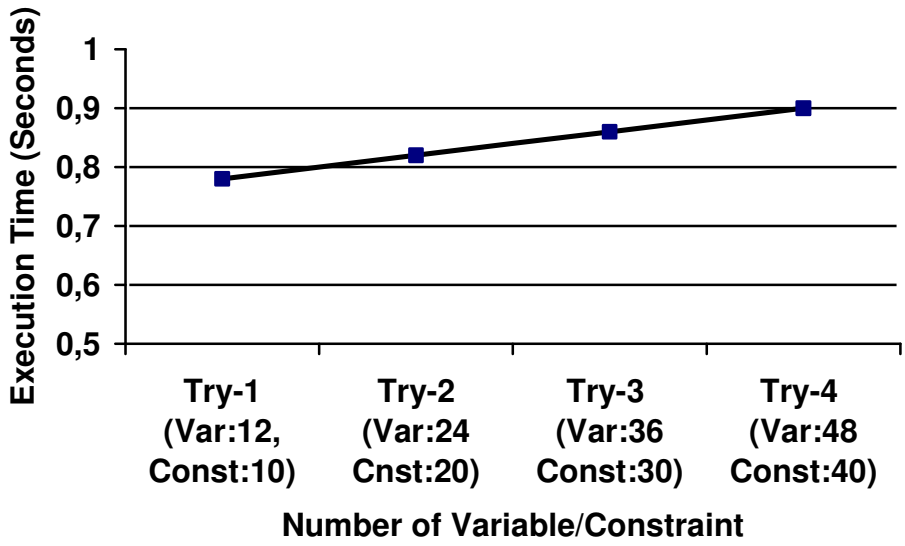

Figure 3.21 Execution time versus Number of Variable/Constraint

As in the other experiments, execution times increases nearly linear when we raise the constraint and variable number. In try-1, execution time is 0, 78 seconds, and in the last try, execution time is 0.90 seconds. In all experiments, solution was found successfully.

3.11.4 Comments on Experiments

As in all experiments shows that affects of number of candidate web services, the number of block structures, and variable/constraint number is below threshold one second. Also, many experiments are complex enough to model web service composition, and all execution times are less than 1 second, and this can be acceptable for user's view. Simplest example's execution takes approximately 0.6 seconds, and most complex example takes 0.9 seconds. This shows that execution time does not increase much with model complexity, and CWSF engine has start-up time approximately 0.5 seconds for each model. Since CSP (Constraint Satisfaction Problem) is NP-Complete problem, it is not possible to say that increase ratio in execution times are polynomial. There can be situations/values in these examples in which constraint engine can perform more searches to find solution.

CHAPTER 4

CWSL DESIGNER

4.1 General Properties

An important contribution of this framework is providing end-users to model their web service composition without requiring extensive knowledge of underlying technology. CWSL Designer also provides guiding system to insert web service templates, and possibility to match ontologies.

Main features of CWSL Designer are as follows;

- Importing and Exporting CWSL files into Designer
- Loading Context OWL files to CWSL problem.
- Ability to design CWSL models with drag-drop facility in graph-based environment.
- Display CWSL block structure in tree navigator
- Defining Semantic Association
- Create, Update, Delete, List Query Constraints
- Create, Update, Delete, List Variables, Constraints (Boolean, IfThen, Execute, ..)
- Create, Update, Delete, List Process Variables, Business Rules
- Define Or, And, Xor, Sequence, Iterate, Decision, Service Template Blocks
- Structured View of Block Hierarchies in graph
- Transition support between blocks in graph
- Running Models to generate solutions, view possible solutions in structured view with displaying selected web service candidates for templates.

In the following sub-sections, general structure and usage of CWSL Designer is illustrated and shortly explained;

4.2 Parts of CWSL Designer

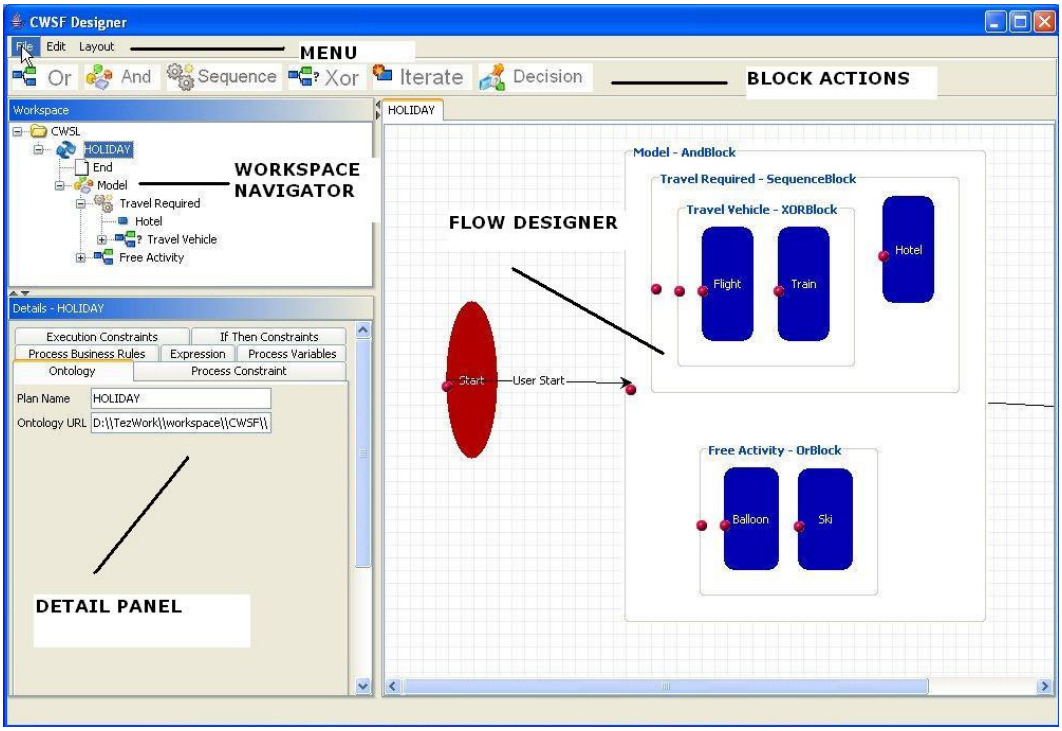

Figure 4.1 Main Parts of CWSL Designer

As shown in Figure 4.1, CWSL designer has five main parts;

- ***Menu:*** A set of actions are available for user interactions such as loading and saving CWSL models, running models for gathering solutions.
- ***Block Actions Panel:*** This panel provides easy-way of making Sequence, Xor, Or, And, Iterate, Decision blocks with combining selected components in the flow designer.
- ***Workspace Navigator:*** Navigator provides tree-based view and access to CWSL Model. Blocks are displayed in tree and selection in tree triggers to

view block properties in the detail panel, and block selection on the flow designer.

- ***Detail Panel:*** All information about blocks is taken from user in this part. For each block, there exist different information panels for manipulating data. Associated sub-panels, tabs, option pane and interaction dialogs exist for low-level parts of the model such as process variables, constraints, business rules.
- ***Flow Designer:*** In this panel, blocks are managed with their sub and parent blocks via drag&drop capabilities in graph-based environment, transitions can be drawn between blocks, creation and deletion of blocks are available.

CWSL Designer produces two files; one is complete CWSL model that can be executed by CWSF Engine and a layout file that configures how to display blocks/transitions in the flow designer. Designer uses open source JGraph [62] library for drawing blocks and transitions.

CWSL Designer initiates itself from a configuration file named as cwsl.properties. In this file, a number of properties are defined such as default directory where CWSL Designer looks first for CWSL definitions, list of integrated context ontologies, etc.

4.3 Usage

In the following sections, usage of CWSL Designer is shortly explained.

4.3.1 Menu Actions

In file menu in Figure 4.2, a number of actions are available such as loading and saving models, running models to get solutions.

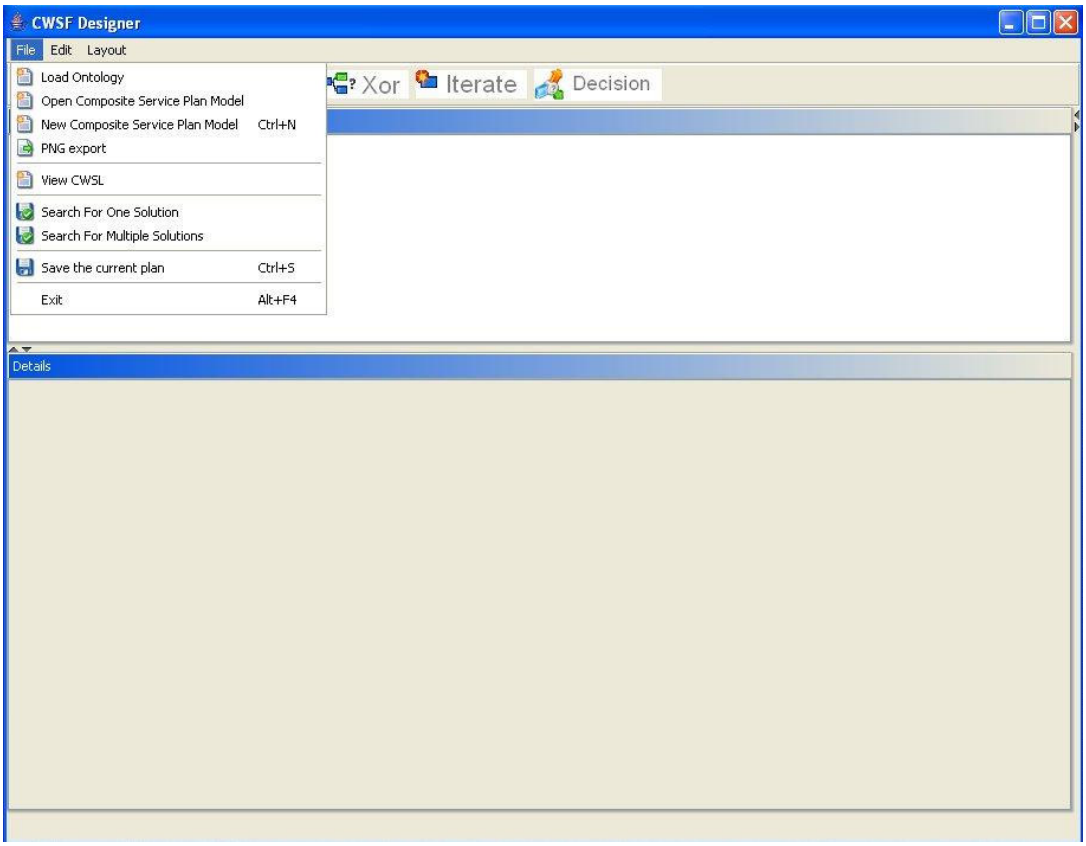

Figure 4.2 Menu Actions

4.3.2 Create New CWSL File

To create new CWSL model, user click “New Composite Web Service Model” item in menu panel in Figure 4.3, and enters the name of the new model. Any time user can use “Save Current Plan” item to save any progress in the model.

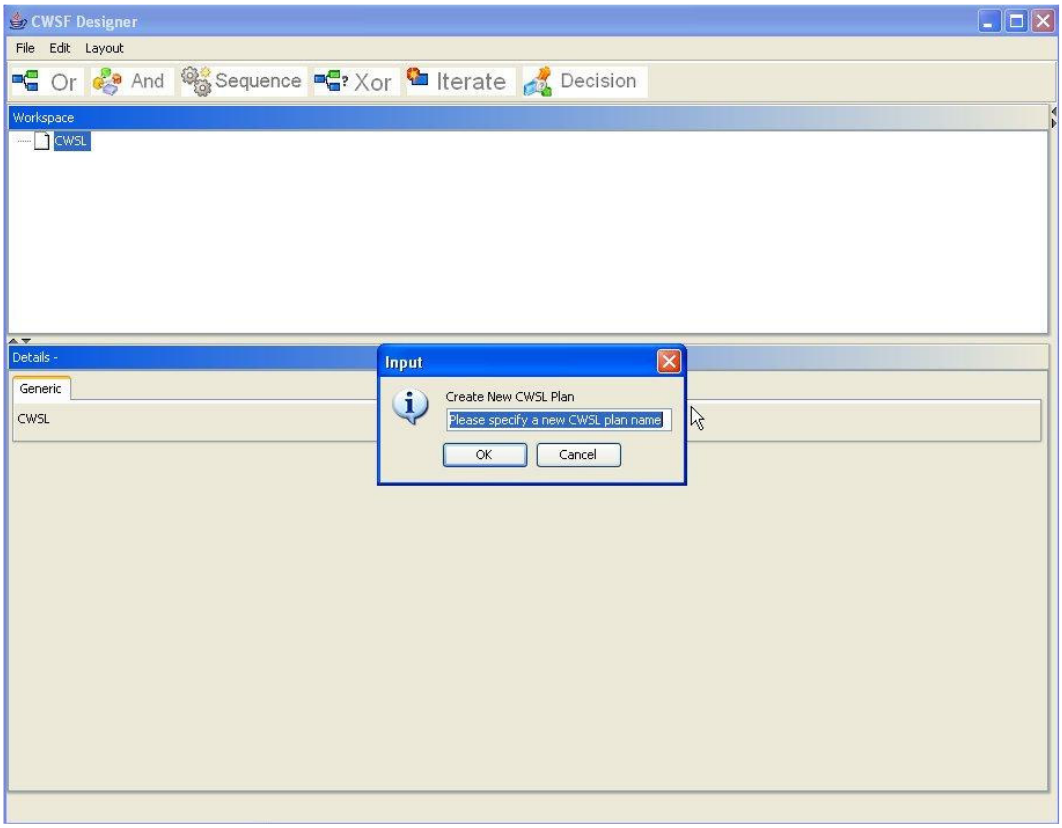

Figure 4.3 Create New CWSL Model

4.3.3 Load CWSL File

As displayed in Figure 4.4, user can reload existing saved model from "Open Composite Web Service Model" item in menu panel. As a requirement, layout information of CWSL model should exist in the same directory.

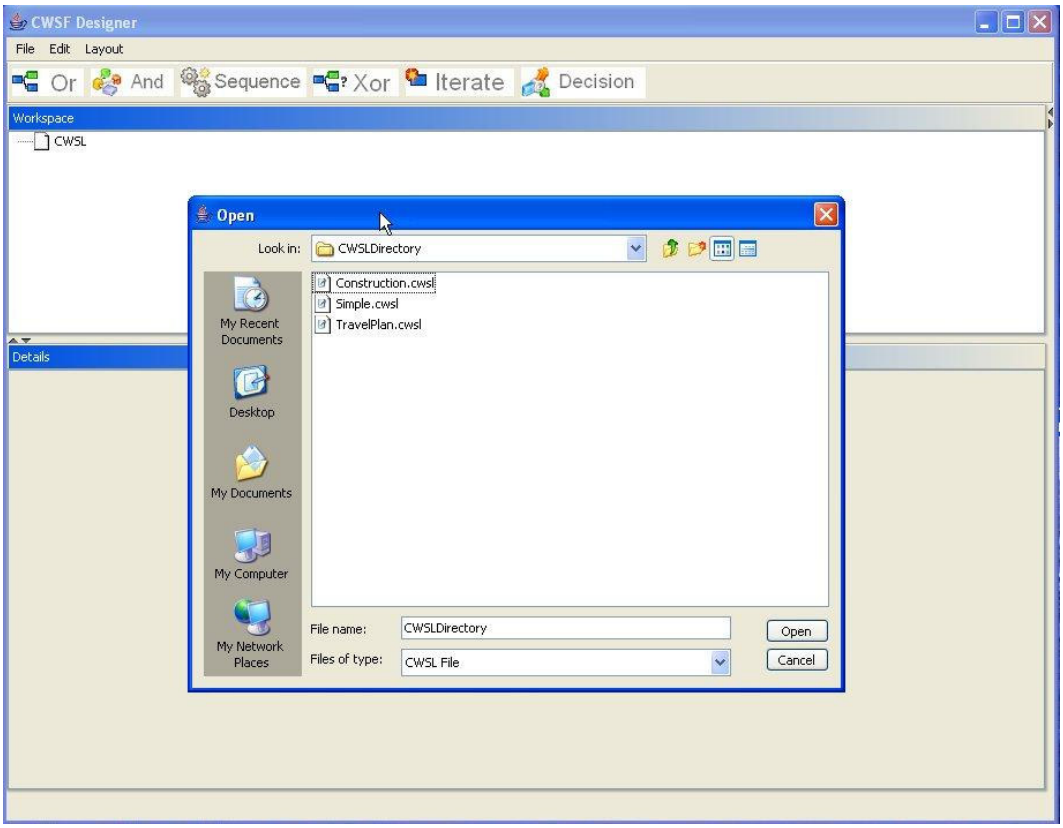

Figure 4.4 Load CWSL Model

4.3.4 Blank CWSL File

In Figure 4.5, a blank CWSL file screen is displayed. When a user creates new CWSL model, a model with only start block is instantiated.

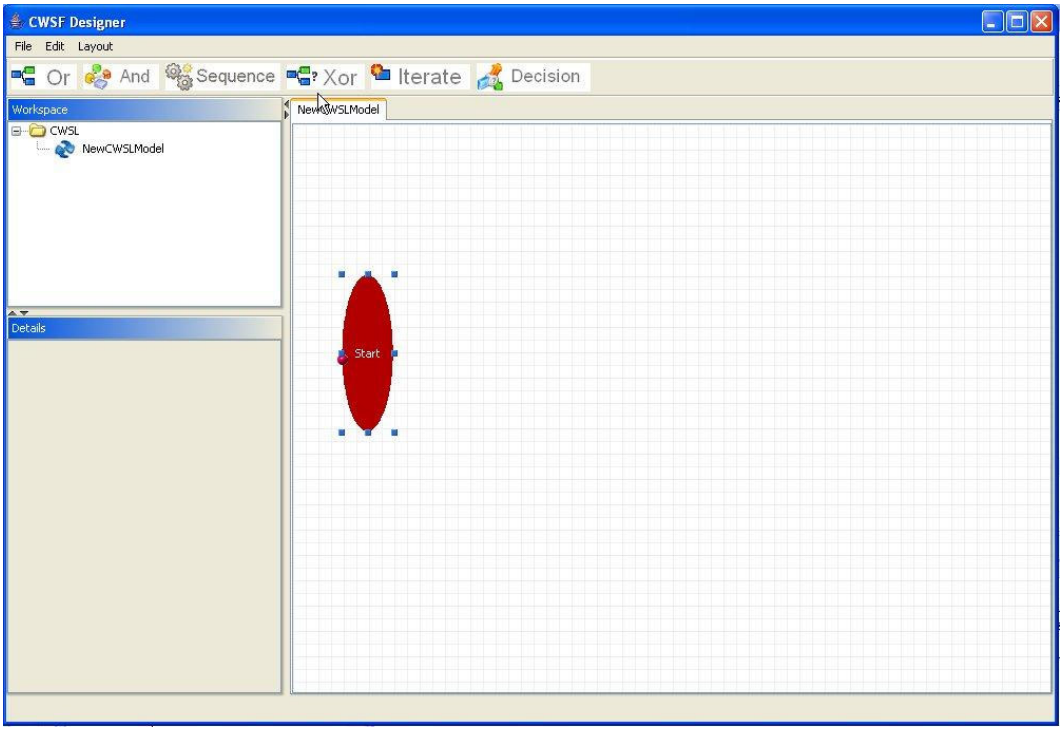

Figure 4.5 Instantiated View of CWSL Model

4.3.5 Load Ontology File

As in Figure 4.6, user can add new context ontologies to CWSF engine using “Load Ontology” item in menu. These ontologies are saved on cwsl.properties file.

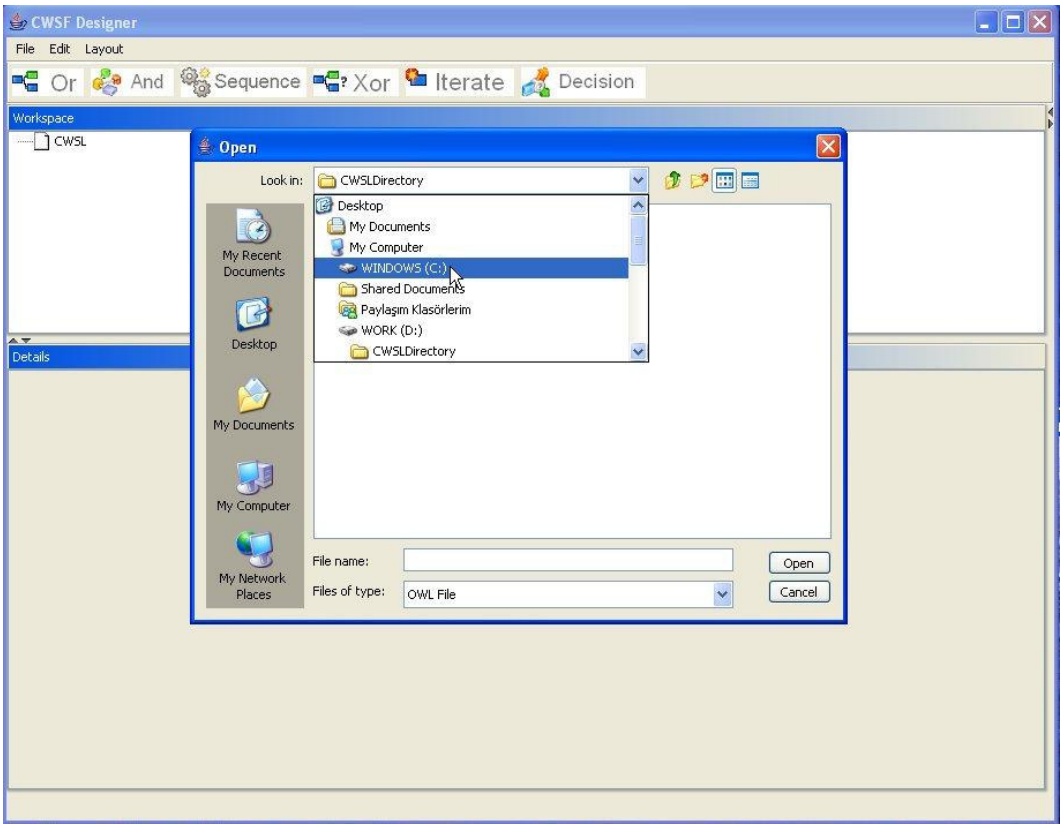

Figure 4.6 Load Context Ontologies

4.3.6 Create Block

As in Figure 4.7, flow designer opens menu popup when user right clicked on it. In that popup, user can create new blocks such as service templates or end blocks.

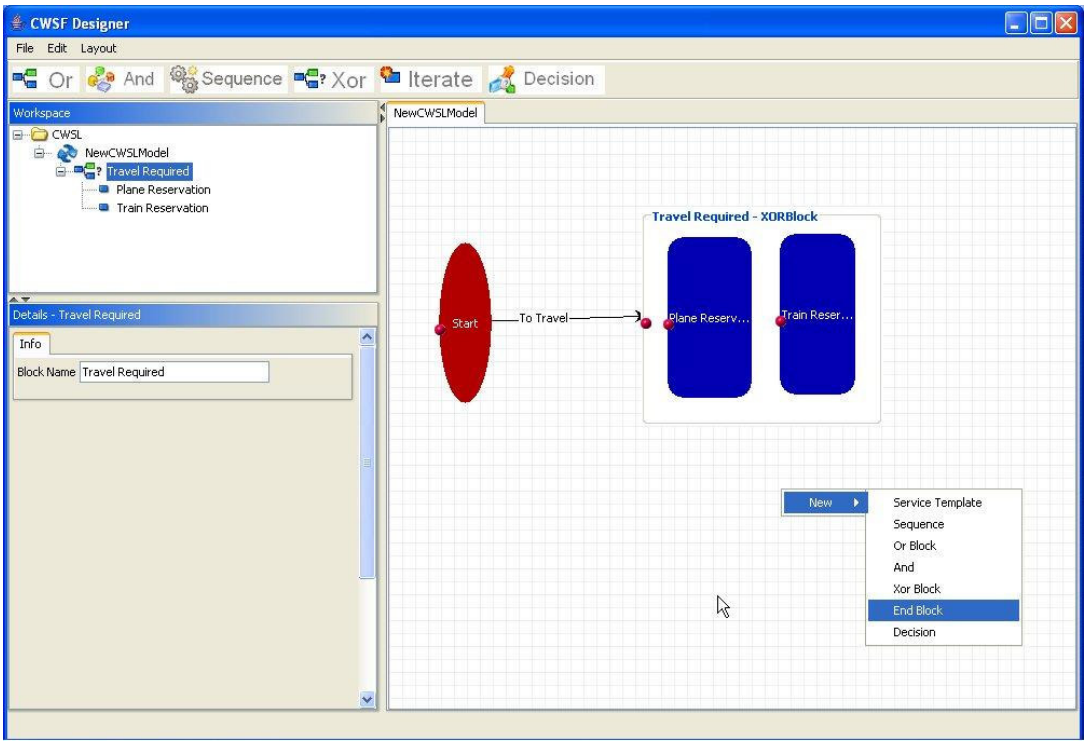

Figure 4.7 Create Blocks

4.3.7 Associating Service Template with Context Ontology

When creating a new service template, it is necessary to select an action in context ontology. As illustrated in Figure 4.8, context ontology is displayed to user as a tree, and user select an action in the referred domain in context.

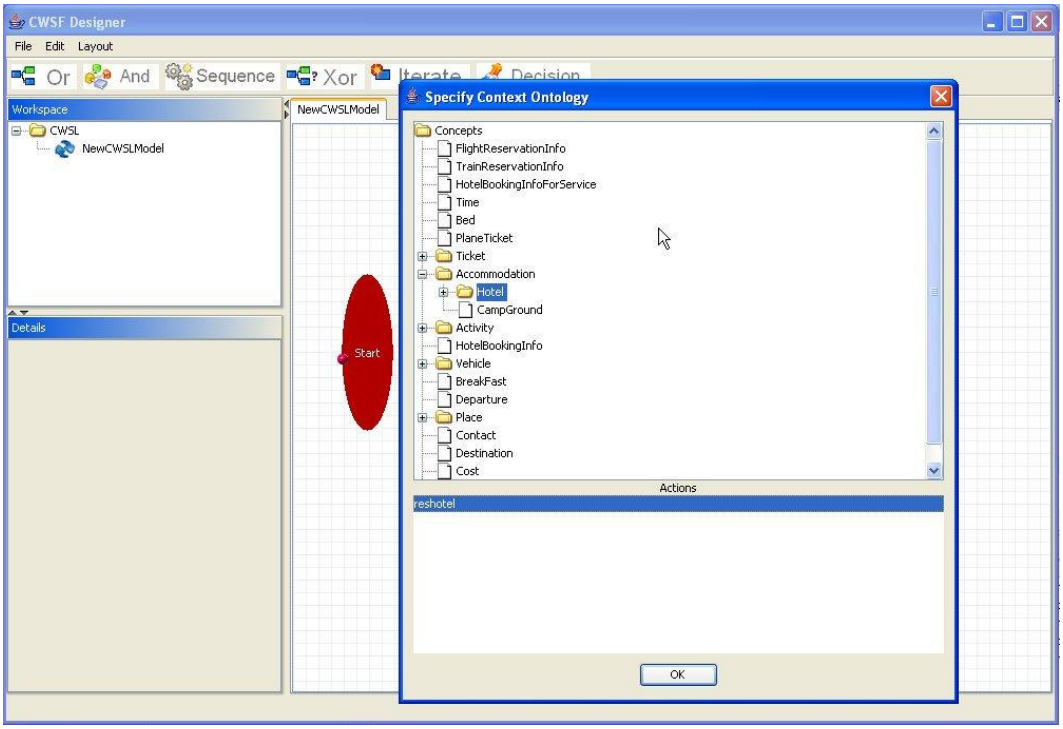

Figure 4.8 Associating Service Template with Context Ontology

4.3.8 Service Template Properties

After selecting a service template as in Figure 4.9, user can control name of the service template, action ontology, query constraints, constraint variables and boolean constraints.

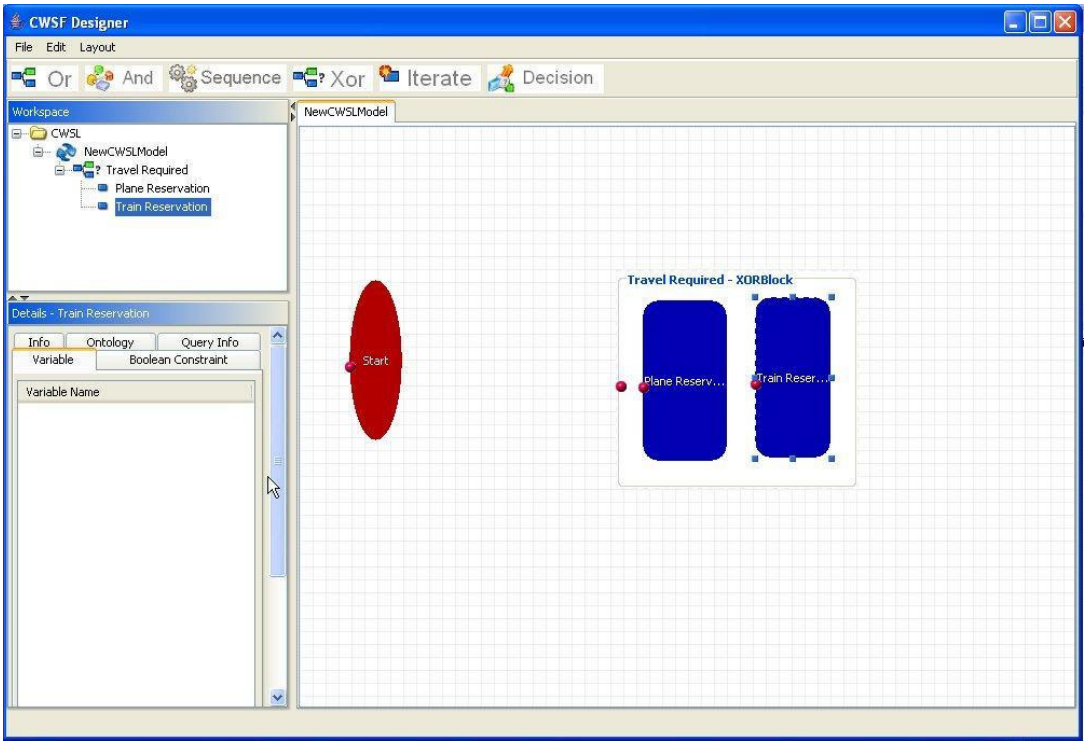

Figure 4.9 Modify Service Template Properties

4.3.9 Create Variable

In Service template detail panel in Figure 4.10, user can create constraint variables with assign method which is either minimize or maximize. Name attribute refers the related context ontology attribute.

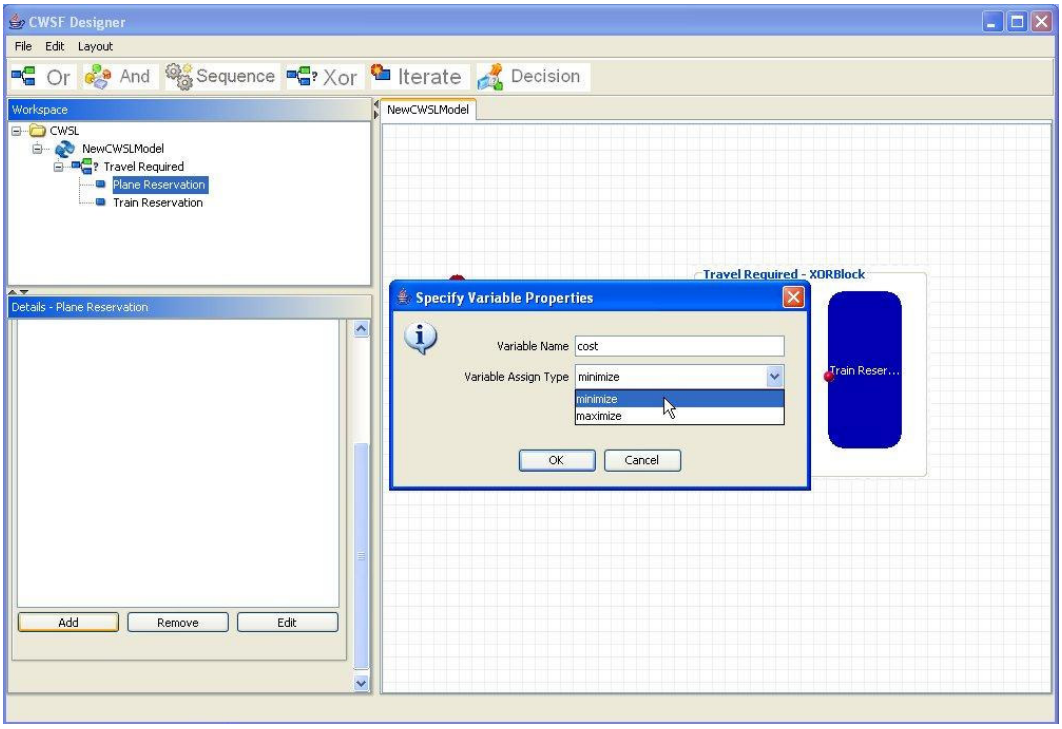

Figure 4.10 Create Variables

4.3.10 Query Constraints

When querying web service candidates, query constraints should be filled for efficient candidate selection. Based on the query ontology, user choose query attribute, related action (equal, not equal, less than, bigger) and defines the value that will be compared against the query context attribute. In figure 4.11, query constraint screen is displayed.

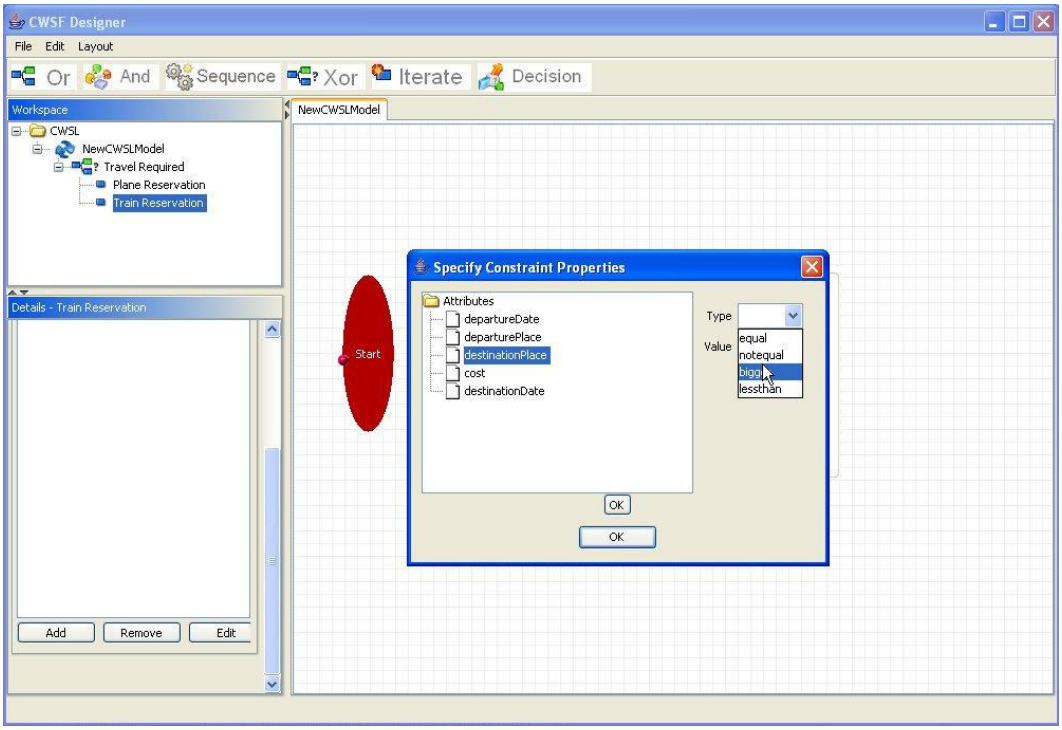

Figure 4.11 Query Constraints for Service Template

4.3.11 Modify CWSL Model Properties

As in Figure 4.12, a set of actions are available to change following items in CWSL model;

- Change process name, and context ontology
- List/Create/Update/Delete Process Variables
- List/Create/Update/Delete Process Constraints
- List/Create/Update/Delete Expressions
- List/Create/Update/Delete Business Rules
- List/Create/Update/Delete If/Then Constraints
- List/Create/Update/Delete Execution Constraints

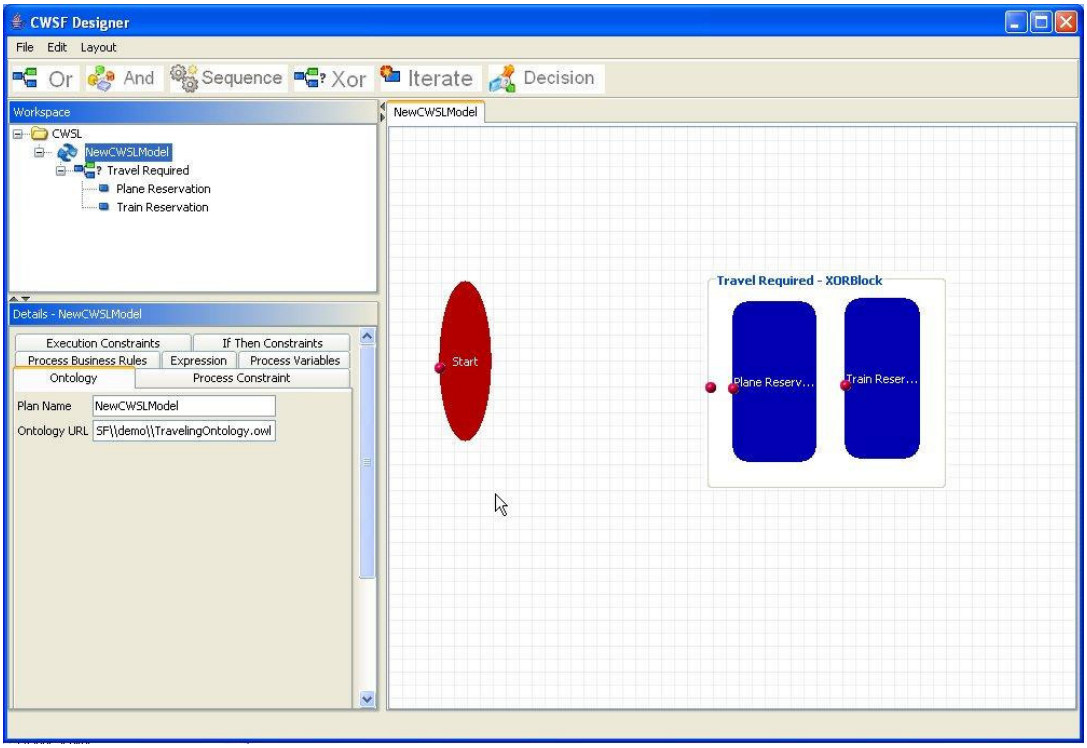

Figure 4.12 Modify CWSL Model Properties

4.3.12 Defining Process Constraint

In CWSL Detail Panel in Figure 4.13, user defines complex constraints based on the expressions, variables and atomic values.

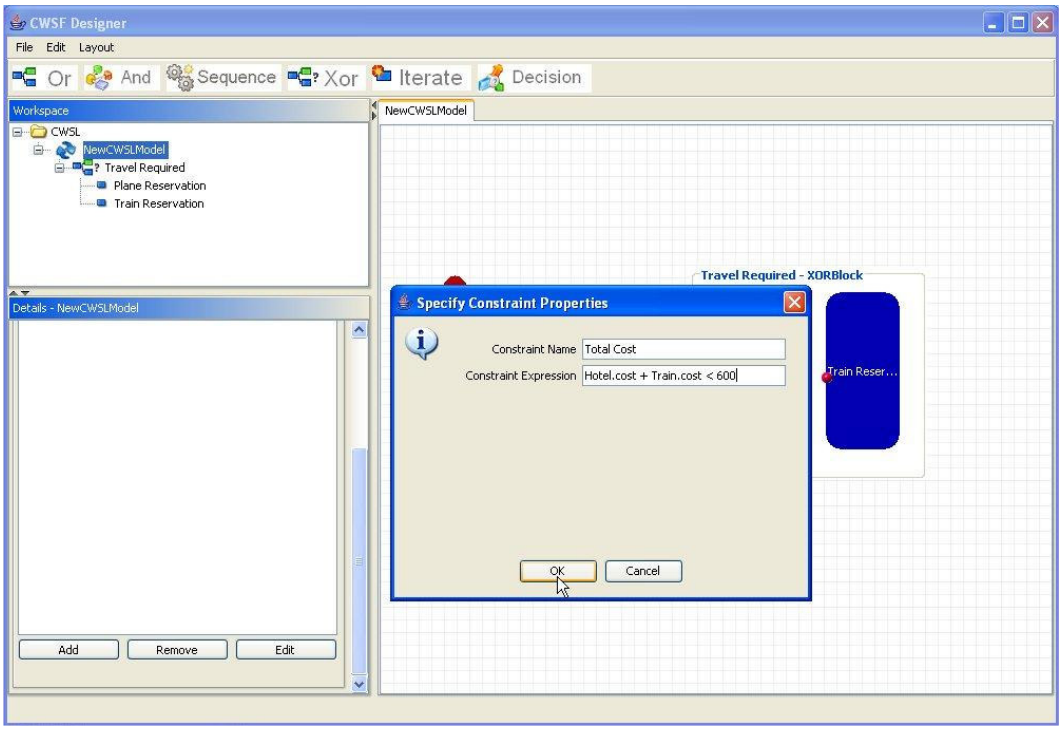

Figure 4.13 Defining Process Constraints

4.3.13 Define Expression

Expressions can be defined similarly in CWSL Details Panel. Name and expression content is required. Expressions are illustrated in Figure 4.14.

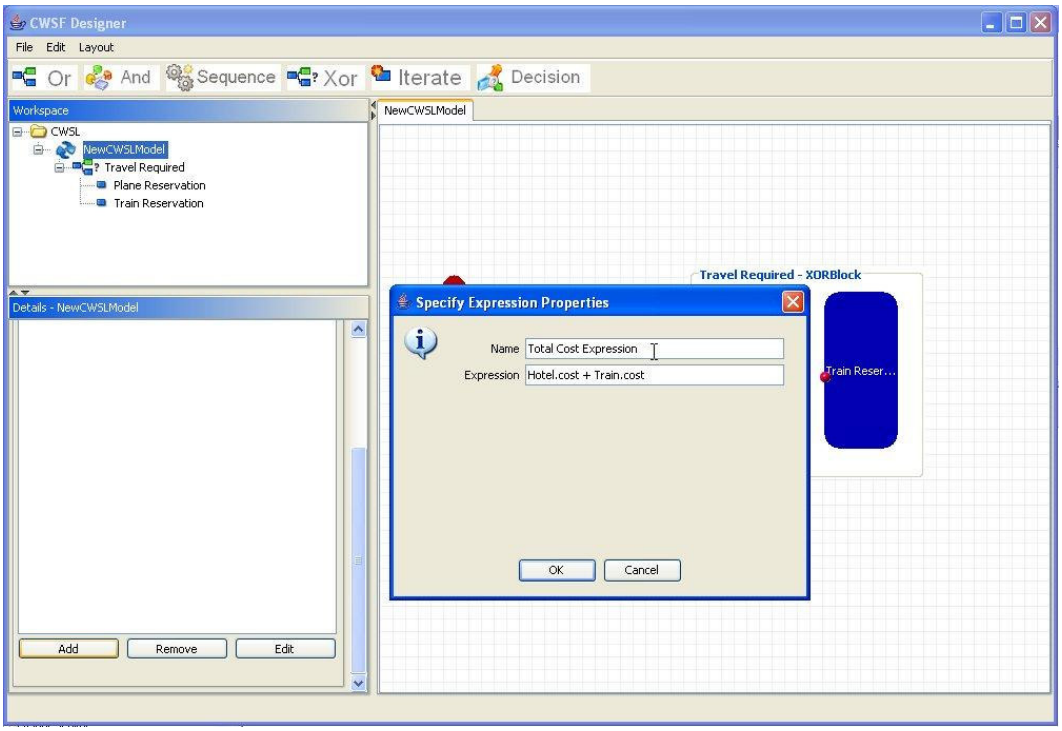

Figure 4.14 Specifying Expression Properties

4.3.14 Add Execute Status Constraint

Execution Status Constraints can be added in CWSL Detail Panel, service template and execution status should be selected. Expressions are illustrated in Figure 4.15.

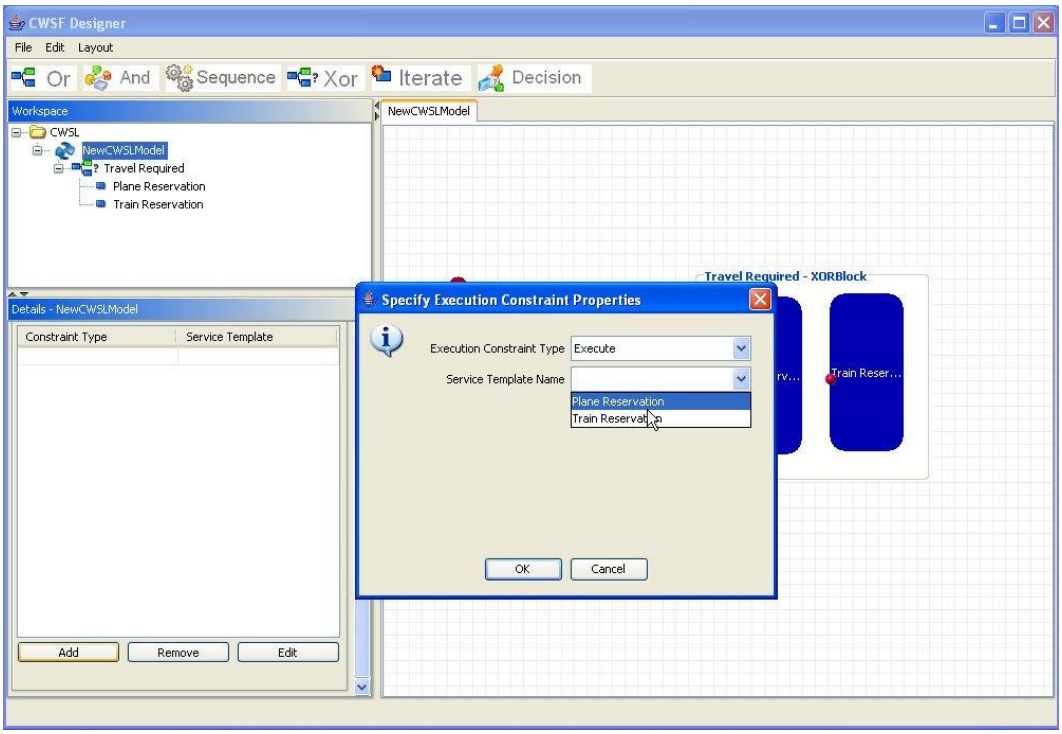

Figure 4.15 Adding New Execution Constraint Properties

4.3.15 Define Process Variable

Process variables can be managed in CWSL Detail Panel as in Figure 4.16. Users should indicate the type of variable, initial value and name of the process variable.

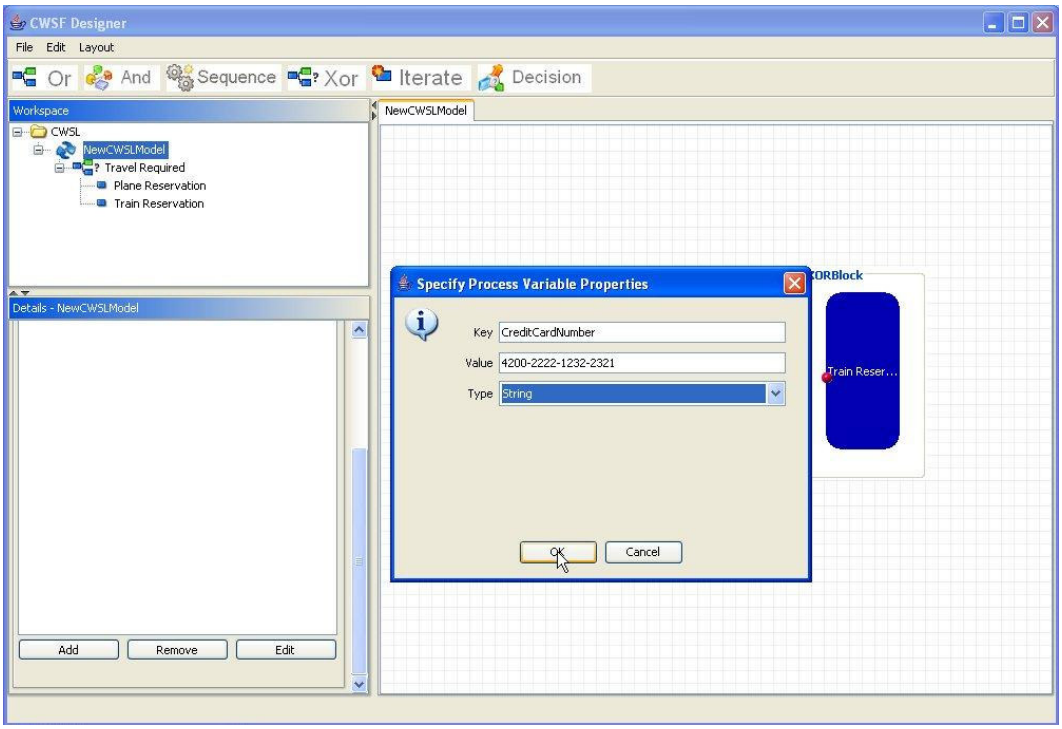

Figure 4.16 Specifying Process Variable Properties

4.3.16 Defining If Then Constraint

If/Then constraint defined in CWSL Detail Panel in Figure 4.17, user should provide name, if constraint, and then constraint.

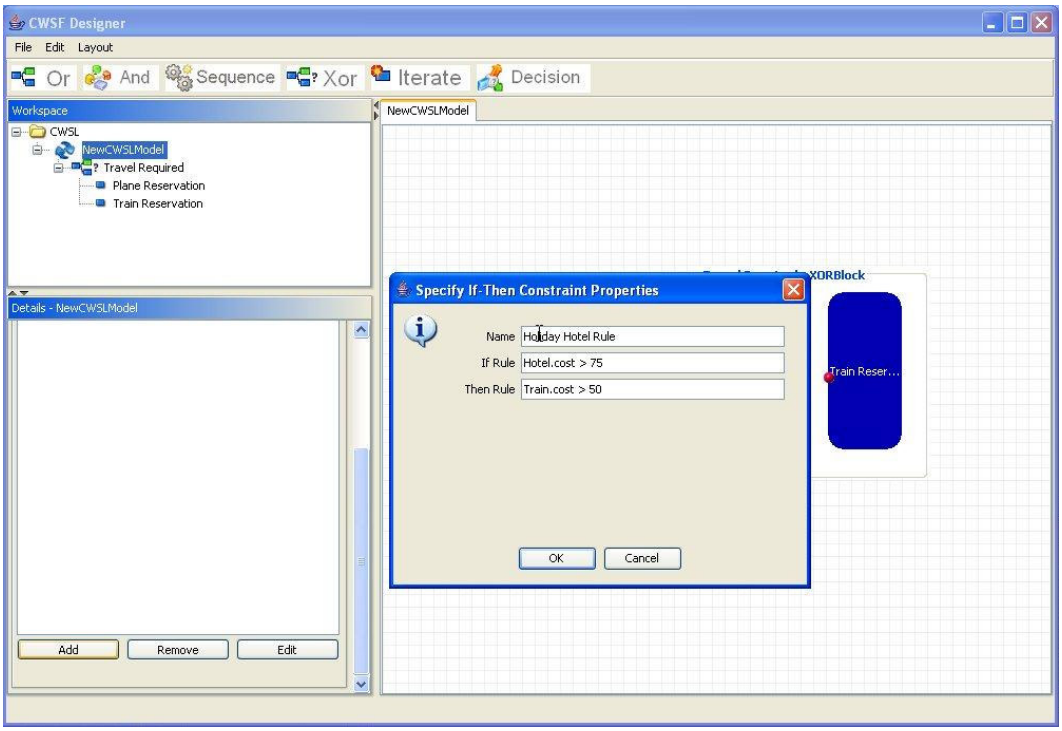

Figure 4.17 Specify If-Then Constraints

4.3.17 Delete Block

An element in the graph can be deleted when right-clicked over block or transition. If it has sub-blocks, these blocks associated with the deleted block's parent. This is illustrated in Figure 4.18.

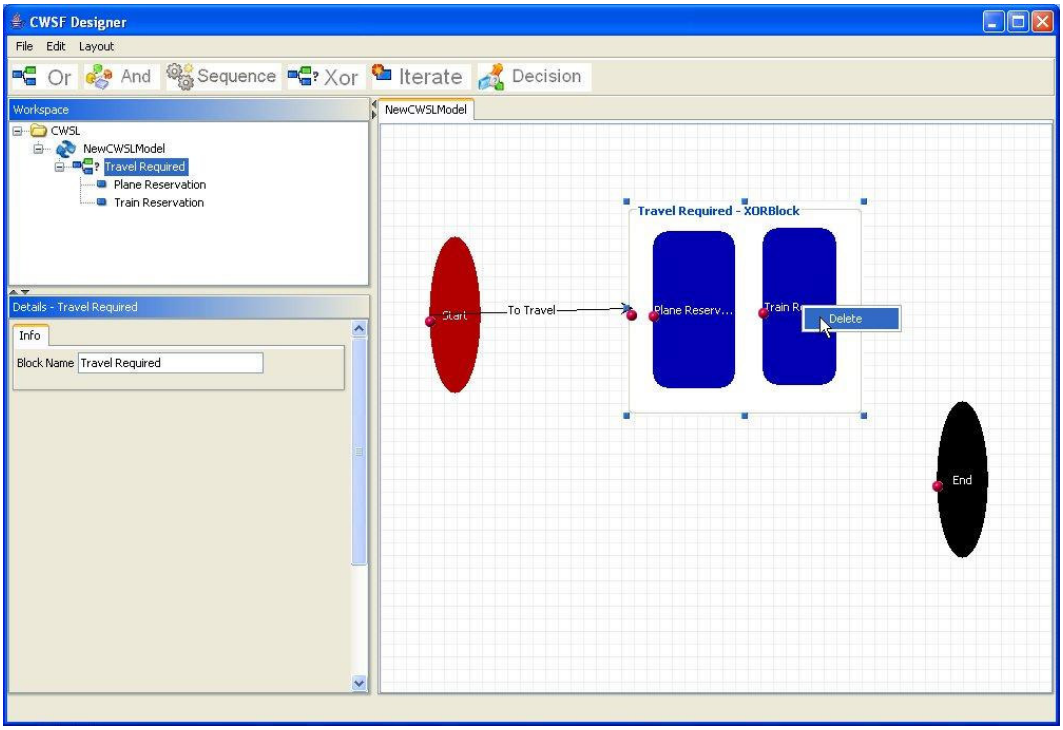

Figure 4.18 Deletion of Blocks or Transitions

4.3.18 Displaying Multiple Solutions

Since CWSF engine can compute multiple solutions. Schedule designer shows possible solutions in a graph with selected service candidates. In the designer, it is possible to view constraint engine problem and solution, and ability to view all other solutions via “previous solution”/”next solution” buttons. This is illustrated in Figure 4.19.

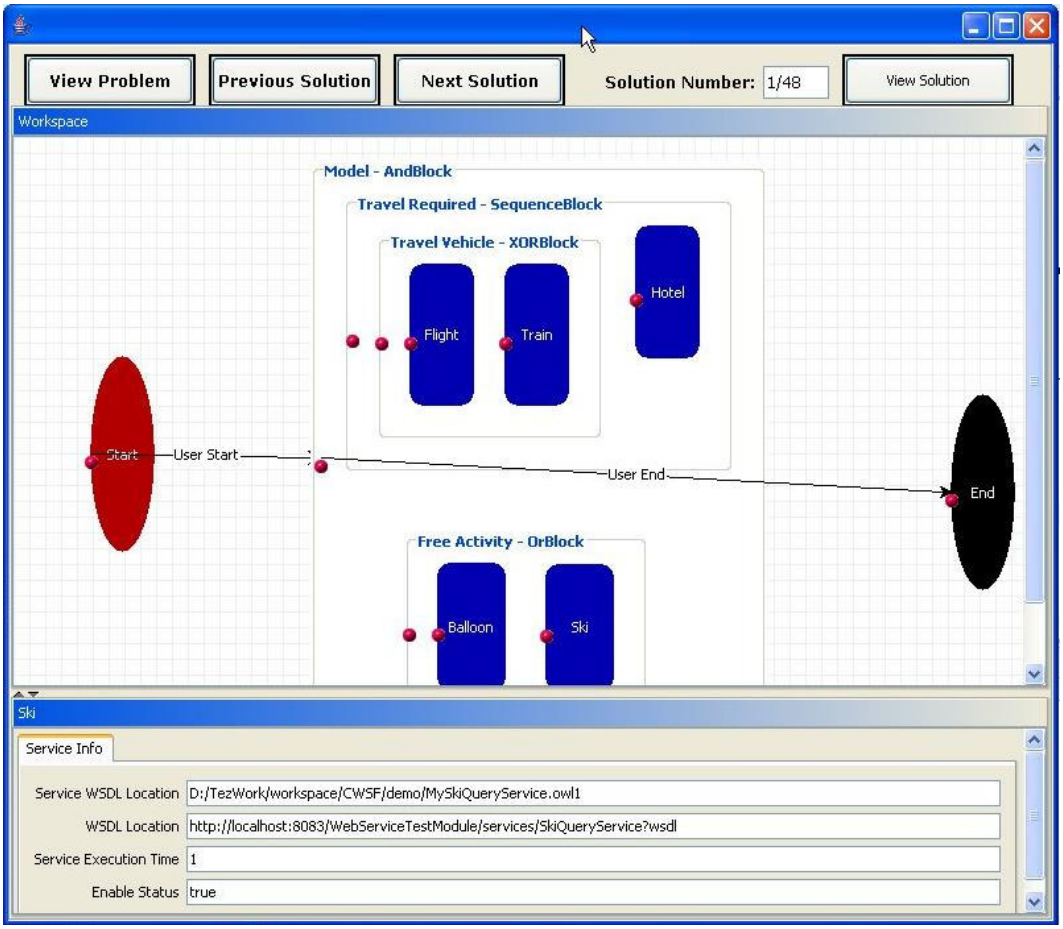

Figure 4.19 Multiple Solutions of CWSF Engine

4.3.19 View Engine Solution

Constraint engine assigns values to variables in the solution. Therefore, user wants to see the solution in detail. This is illustrated in Figure 4.20.

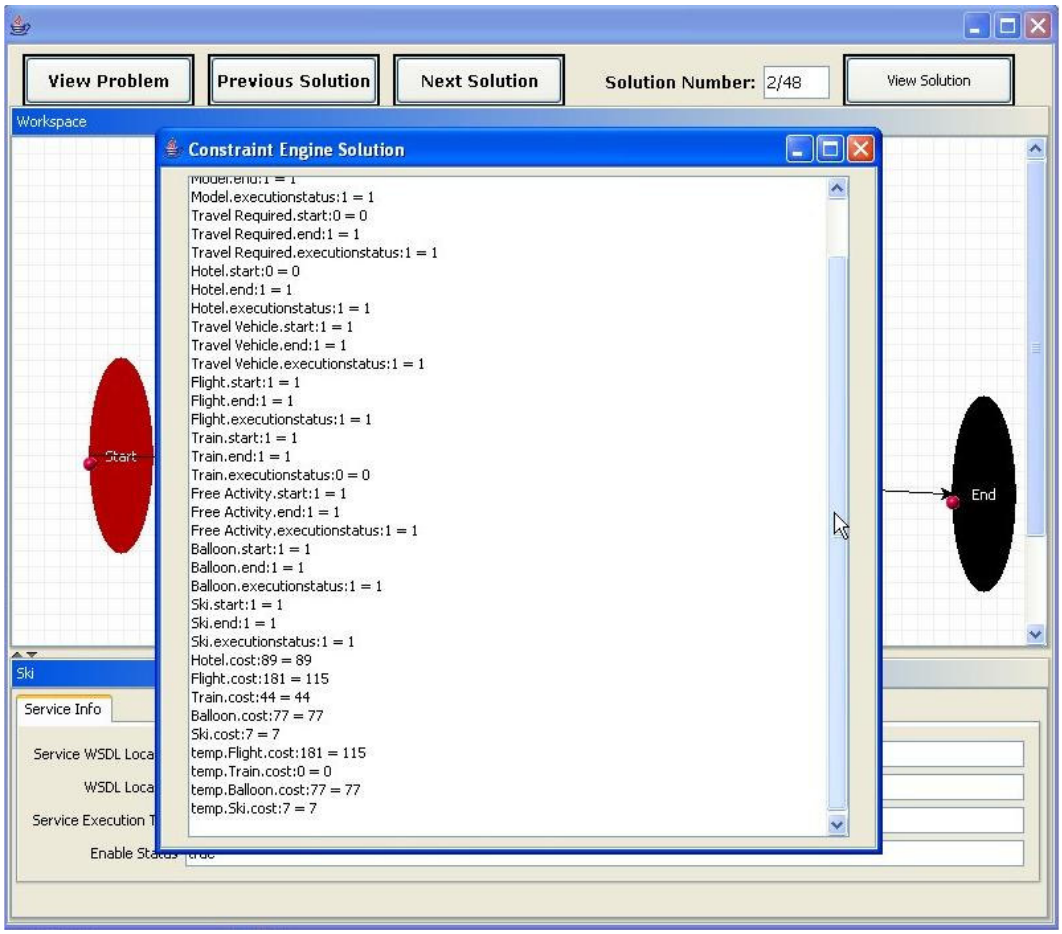

Figure 4.20 Engine Solution

CHAPTER 5

CONCLUSION

In this thesis, a web service composition system named as CWSF (Composite Web Service Framework) is constructed for scheduling of web services under resource allocation constraints. Motivation comes from increasing rate of available web services every day, and it is not possible for ordinary users to search, select and compose web services from a huge web service repository where many user constraints are defined. Also, one can not expect to discover and check the semantically and syntactically correct web service composition under the rules that user define. Current standards focus on statically defined composition systems where web services are already defined and integrated, no dynamic selection of web services are available. Therefore, an automated framework is needed to discover web services on the fly, define composition constraints to select web service to optimize solution and schedule web services in a correct order that guarantees correct execution without a failure on execution time. CWSL (Composite Web Service Language) defined for describing mainly workflow structure, web service templates and constraints. CWSF converts web service composition problem into constraint satisfaction problem to find solutions. An external Constraint engine Choco is utilized for the evaluation of constraints.

Mainly, the power of CWSF comes with the dynamism of defining and planning of composition, main contribution of CWSF is to select and integrate web services into executable schedules under a large set of constraints, and provide minimum-cost solutions. CWSF setups a workflow language supporting nested block structure with many types of block structure, rich set of resource allocation constraints (cost, control, ordering, logical constraints and business rules) on composition, semantic

discovery of web services using UDDI and Query Structures for plan generation. Also, CWSF converts Web Service Composition Problem into Constraint Satisfaction Problem using developed constraint translator. User-friendly design with guiding system orients users to define composition and displays other possible solutions to provide users flexibility for selection of other executable schedules.

With this work, it is shown that Constraint Programming approach can be used for solving web service composition problems. This approach can be used in many real life problems, and experiments show that CWSF is efficient enough to model these problems.

As a future work, CWSF can be integrated with standards such as BPEL to directly invoke execution process. Also, since discovering and querying web services requires important time, a method can be setup to save models templates, and caching the results of web services to reduce scheduling time.

REFERENCES

[1] B. Benatallah, M. Dumas, M. C. Fauvet and F.A. Rabhi, 2002. Towards Patterns of Web Services Composition. In Gorlatch, S. and F. Rabhi (Eds.), Patterns and Skeletons for Parallel and Distributed Computing. Springer Verlag (UK).

[2] B. Benatallah, M Dumas, Q Sheng , Facilitating the rapid development and scalable orchestration of composite Web services, Distributed and Parallel Databases, pp. 5 – 37, 2005.

[3] Universal description, discovery and Integration of Web Services, http://www.uddi.org/, April 2007

[4] F. Curbera., W. Nagy and S. Weerawarana, Web Services: Why and how. Workshop on object oriented web services. OOPSLA Tampa, Florida, USA, 2001.

[5] BPEL4WS, Business Process Execution Language for Web Services, ftp://www6.software.ibm.com/software/developer/library/ws-bpel11.pdf, April 2007

[6] OWL-S: Semantic Markup for web services. OWL white paper, http://www.daml.org/services/owls/1.0/owl-s.pdf, April 2007

[7] D. Fensel, and C. Bussler, 2002. Semantic web enabled web services. In Proceedings of International Semantic Web Conference (ISWC'2002), volume 2342.

[8] A. Lazovik, M. Aiello, and R. Gennari, Encoding Requests to Web Service Compositions as Constraints. (2005) LNCS 3709:782-786. *Constraint Programming (CP) 2005* Springer.

[9] Choco constraint programming system, http://choco.sourceforge.net/, April 2007

[10] S. A. McIlraith, T. C. Son: Adapting Golog for Composition of Semantic Web Services. KR 2002: 482-496.

[11] B. Medjahed, A. Bouguettaya, A.K. Elmagarmid , Composing Web services on the Semantic Web, VLDB J. 12(4): 333-351, 2003.

[12] M. Paolucci, T. Kawamura, T. R. Payne, K. Sycara, Importing the Semantic Web in UDDI, WES 2002: 225-236.

[13] M. Paolucci, T. Kawamura, T. R. Payne, K. Sycara, Semantic Matching of Web Services Capabilities, International Semantic Web Conference, 333-347, 2002.

[14] Web Service Description Language, WSDL, http://www.w3.org/TR/wsdl, April 2007

[15] J. Rao and X. Su, A Survey of Automated Web Service Composition Methods, SWSWPC, 43-54, 2004.

[16] P. Senkul, M. Kifer, I Toroslu, A logical Framework for Scheduling Workflows under Resource Allocation Constraints, VLDB 2002.

[17] Pinar Senkul, Ismail Hakki Toroslu: An architecture for workflow scheduling under resource allocation constraints. Inf. Syst. 30(5): 399-422., 2005.

[18] Pinar Senkul: CompositeWeb Service Construction by Using a Logical Formalism. ICDE Workshops 2006: 56, 2006
[19] E. Sirin, B Parsia, D .Wu, J. Hendler, D. Nau, HTN Planning for Web Service Composition Using SHOP2, Journal of Web Semantics, 1(4): 377-396, 2004.
[20] Workflow Management Coalition. Terminology and glossary ver 3.0. Technical Report (WFMC-TC-1011), Workflow Management Coalition, Brussels, February 1999.
[21] D. Box et al. Simple Object Access Protocol (SOAP) 1.1. Online:http://www.w3.org/TR/SOAP/, April 2007
[22] R. Chinnici et al. Web Services Description Language (WSDL) 1.2. Online: http://www.w3.org/TR/wsdl/, April 2007
[23] D. Martin et al. DAML-S(and OWL-S) 0.9 draft release, http://www.daml.org/services/daml-s/0.9/, April 2007
[24] 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.
[25] T. Andrews, Business Process Execution Language for Web Services (BPEL4WS) 1.1, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel, April 2007
[26] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid. Composing Web services on the Semantic Web. The VLDB Journal, 12(4), November 2003.
[27] F. Casati, S. Ilnicki, and L. Jin. Adaptive and dynamic service composition in EFlow. In Proceedings of 12th International Conference on Advanced Information Systems Engineering (CAiSE), Stockholm, Sweden, June 2000. Springer Verlag.
[28] F. Casati, M. Sayal, and M.-C. Shan. Developing e-services for composing eservices.In Proceedings of 13th International Conference on Advanced Information Systems Engineering (CAiSE), Interlaken, Switzerland, June 2001. Springer Verlag.
[29] S. McIlraith and T. C. Son. Adapting Golog for composition of Semantic Web services. In Proceedings of the 8th International Conference on Knowledge Representation and Reasoning (KR2002), Toulouse, France, April 2002.
[30] S. McIlraith, T. C. Son, and H. Zeng. Semantic Web services. IEEE Intelligent Systems, 16(2):46–53, March/April 2001.
[31] S. Narayanan and S. McIlraith. Simulation, verification and automated composition of Web service. In Proceedings of the 11th International World Wide Web Conference, Honolulu, Hawaii, USA, May 2002.
[32] E. Sirin, J. Hendler, and B. Parsia. Semi-automatic composition of Web services using semantic descriptions. In Proceedings of Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS2003, 2002.
[33] 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.
[34] E. C. Freuder, Constraints, 1997

[35] Choco User Guide, http://choco-solver.net/index.php?title=User_guide, April 2007
[36] N. Milanovic and M. Malek, Current solutions for Web service composition, IEEE Internet Computing, 2004
[37] Web Service Specifications, http://en.wikipedia.org/wiki/List_of_Web_service_specifications, April 2007
[38] Alonso, G., Casati, F., Kuno, H. and Machiraju, V. (2004) Web Services. Concepts, Architectures and Applications, Springer-Verlag Berlin Heidelberg.
[39] Doulkeridis, C., Valavanis, E. and Vazirgiannis, M. (2003) Benatallah, B. and Shan, M-C. (Eds.): Towards a Context-Aware Service Directory, TES, LNCS 2819, Springer-Verlag Berlin Heidelberg, pp.54–65.
[40] Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. and Orchard, D. (2004) Web Services Architecture, W3C Working Group Note 11, February, W3C Technical Reports and Publications, http://www.w3.org/TR/ws-arch/, April 2007
[41] Schahram Dustdar and Wolfgang Schreiner, A survey on web services composition, Int. J. Web and Grid Services, Vol. 1, No. 1, 2005
[42] IBM WebSphere, http://www-306.ibm.com/software/websphere/, April 2007
[43] Oracle BPEL Process Manager, http://www.oracle.com/technology/products/ias/bpel/index.html, April 2007
[44] Microsoft BizTalk Server, http://www.microsoft.com/biztalk, April 2007
[45] Active BPEL, http://www.activebpel.org, April 2007
[46] RDF, Resource Description Framework, http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/, April 2007
[47] OWL, Web Ontology Language, http://www.w3.org/2004/OWL/, April 2007
[48] Web Service, http://en.wikipedia.org/wiki/Web_service
[49] S. Narayanan and S. McIlraith, “Simulation, Verification and Automated Composition of Web Services,” Proc. Int’l World Wide Web Conf. (WWW2002), 2002, pp. 77–88.
[50] XML, Extensible Markup Language , http://en.wikipedia.org/wiki/XML, April 2007
[51] UDDI4J, http://uddi4j.sourceforge.net/, April 2007
[52] JUDDI, http://ws.apache.org/juddi/, April 2007
[53] BSF, Bean Scripting Framework, http://jakarta.apache.org/bsf/, April 2007
[54] Java, http://java.sun.com/, April 2007
[55] Javascript, http://en.wikipedia.org/wiki/JavaScript, April 2007
[56] Python, http://www.python.org/, April 2007
[57] Groovy, http://groovy.codehaus.org/, April 2007
[58] E. Karakoc, K. Kardas, P. Senkul, A Workflow-Based Web Service Composition System, 2006 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT 2006 Workshops)(WI-IATW'06)
[59] SOAP, http://en.wikipedia.org/wiki/SOAP, April 2007
[60] OWL, Web Ontology Language, http://www.w3.org/2004/OWL/, April 2007
[61] J. Klein. Advanced rule-driven transaction management. In IEEE COMPCON.

[62] JGraph, http://www.jgraph.com/, April 2007
[63] OASIS, Organization for the Advancement of Structured Information Standards, http://www.oasis-open.org/home/index.php, April 2007
[64] Aggarwal, R. Kunal Verma Miller, J. Milnor, W., Constraint driven Web service composition in METEOR-S, Services Computing, 2004. (SCC 2004). Proceedings.

APPENDIX A

CWSL EXAMPLES

In this section, complete example is given in the travel domain to illustrate the usage of CWSL.

A.1 Traveling Domain Example

In this example, a sample complete CWSL example in section 3.2 is given including problem and solution definitions. Travel Planner Example is also given in Figure A.1

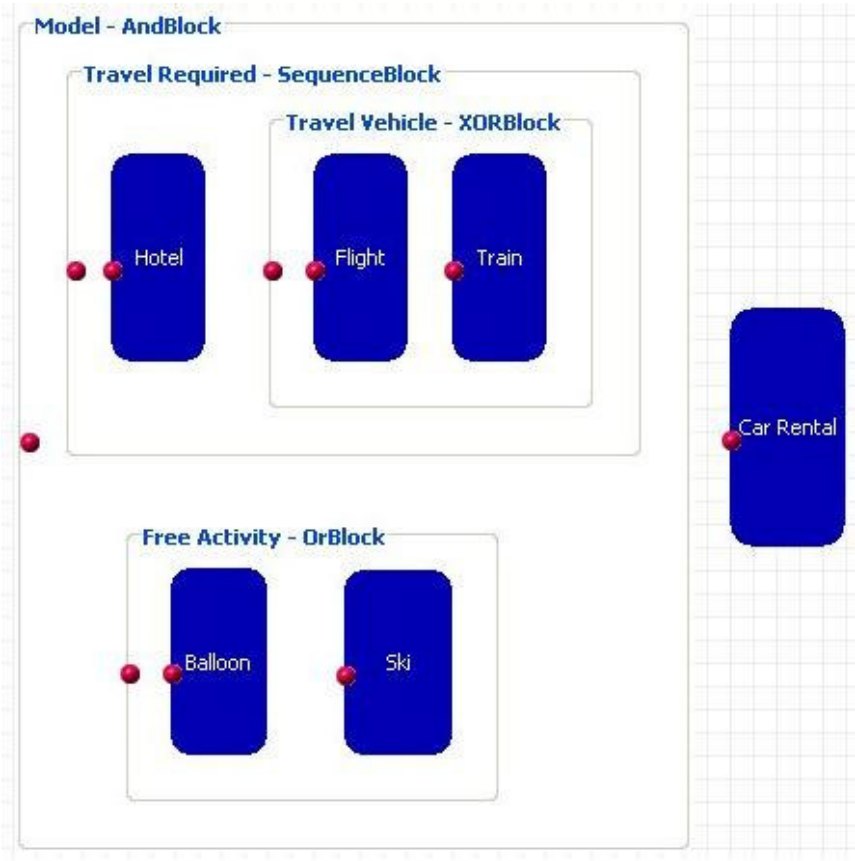

Figure A.1 Travel Planner Example

Below, there is list of constraints defined by user.

- My maximum budget is only 1500 $ for my travel.
- I want to go to “Antalya” for my holiday.
- I want to start my travel on Wednesday (25.04.2007) and finish at Sunday (29.04.2007).
- Distance between hotel location and city center should not be more than 15 km.
- Hotel should have minimum four stars.
- I can give maximum 600 $ for 4 days for my hotel.
- I can give extra 200 $ if hotel is “Hilton”.
- I prefer Plane if it is cheaper than Train.
- Free activities are not mandatory; if they are available then I will be happier.
- In either case, I do not want to spend more than 250 $ for my free activities.
- Car Rental is a must for my travel.
- I want to use a “Toyota” Car during my holiday, no other cars.
- Hotel Company and Car Rental Company should be the same.
- I can pay 100$ gor car rental,.
- I want to go by plane if it is less than 100 $.
- If I go by train, time of travel should not be more than 10 hours.
- My Balloon tour should last at least one hour.
- Ski activity should include an instructor.
- Also if ski activity is selected, then my ski activity should occur at least 2 times.

Full CWSL model of the model is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<cwsl name="HOLIDAY">
    <composite-process>
        <ontology>
            <url>TravelingOntology</url>
        </ontology>
        <sequence name="HOLIDAY">
            <end name="End"/>
            <and name="Model">
                <sequence name="Travel Required">
                    <service-template name="Hotel">
                        <ontology-class>
                            <owlClassModel>Hotel</owlClassModel>
                            <action>reshotel</action>
                        </ontology-class>
                        <variable-set>
                            <variable assign-method="minimize" name="cost" type="Int"/>
                            <variable assign-method="minimize" name="distanceToCityCentre"
type="Int"/>
                            <variable assign-method="minimize" name="location" type="String"/>
                            <variable assign-method="maximize" name="star" type="Int"/>
                            <variable assign-method="minimize" name="numberOfDays" type="Int"/>
                            <variable assign-method="minimize" name="name" type="String"/>
                        </variable-set>
                        <constraint-set/>
                        <query>
                            <criteria/>
                        </query>
                    </service-template>
                    <xor name="Travel Vehicle">
                        <service-template name="Flight">
                            <ontology-class>
                                <owlClassModel>Plane</owlClassModel>
                                <action>resplane</action>
                            </ontology-class>
                            <variable-set>
                                <variable assign-method="minimize" name="cost" type="Int"/>
                                <variable assign-method="minimize" name="from" type="String"/>
                                <variable assign-method="minimize" name="to" type="String"/>
                                <variable assign-method="minimize" name="time" type="String"/>
                            </variable-set>
                            <constraint-set/>
                            <query>
                                <criteria/>
                            </query>
                        </service-template>
                        <service-template name="Train">
                            <ontology-class>
                                <owlClassModel>Train</owlClassModel>
                                <action>resyen</action>
                            </ontology-class>
                            <variable-set>
                                <variable assign-method="minimize" name="cost" type="Int"/>
                                <variable assign-method="minimize" name="from" type="String"/>
                                <variable assign-method="minimize" name="to" type="String"/>
                                <variable assign-method="minimize" name="timeOfTravel" type="Int"/>
                            </variable-set>
                            <constraint-set/>
                            <query>
                                <criteria/>
                            </query>
                        </service-template>
                    </xor>
```

```
            </sequence>
            <or name="Free Activity">
                <service-template name="Balloon">
                    <ontology-class>
                        <owlClassModel>Balloon</owlClassModel>
                        <action>resballoon</action>
                    </ontology-class>
                    <variable-set>
                        <variable assign-method="minimize" name="cost" type="Int"/>
                        <variable assign-method="minimize" name="duration" type="Int"/>
                    </variable-set>
                    <constraint-set/>
                    <query>
                        <criteria/>
                    </query>
                </service-template>
                <service-template name="Ski">
                    <ontology-class>
                        <owlClassModel>Ski</owlClassModel>
                        <action>ressk</action>
                    </ontology-class>
                    <variable-set>
                        <variable assign-method="maximize" name="cost" type="Int"/>
                        <variable assign-method="maximize" name="time" type="String"/>
                        <variable assign-method="maximize" name="instructor" type="String"/>
                    </variable-set>
                    <constraint-set/>
                    <query>
                        <criteria/>
                    </query>
                </service-template>
            </or>
            <transition name="to car" to="CarRental"/>
        </and>
        <service-template name="CarRental">
            <ontology-class>
                <owlClassModel>CarRental</owlClassModel>
                <action>rescarrental</action>
            </ontology-class>
            <variable-set>
                <variable assign-method="maximize" name="cost" type="Int"/>
                <variable assign-method="maximize" name="carType" type="String"/>
                <variable assign-method="minimize" name="motor" type="String"/>
            </variable-set>
            <constraint-set/>
            <query>
                <criteria/>
            </query>
            <transition name="to end" to="End"/>
        </service-template>
    </sequence>
    <business-rules>
        <businessrule name="Hilton Hotel" onexpression="rulexp">
            <if>Hotel.name == "Hilton"</if>
            <then>$rulexp + 200</then>
        </businessrule>
    </business-rules>
    <process-variable-set/>
    <expression-set>
        <expression name="rulexp">
        Hotel.cost + Flight.cost + Train.cost + Balloon.cost + Ski.cost + CarRental.cost
        </expression>
    </expression-set>
```

```
        <process-constraints>
            <constraint name="some">
                $rulexp &lt; 1500
            </constraint>
            <constraint name="some">
                Hotel.location == "Antalya"
            </constraint>
            <constraint name="some">
                Hotel.distanceToCityCentre &lt; 15
            </constraint>
            <constraint name="some">
                Hotel.star &gt; 3
            </constraint>
            <constraint name="some">
                Hotel.numberOfDays == 4 &amp;&amp; Hotel.cost &lt; 600
            </constraint>
            <constraint name="some">
                Balloon.cost + Ski.cost &lt; 250
            </constraint>
            <constraint name="some">
                CarRental.cost &lt; 100 &amp;&amp; CarRental.motor = "1.6" &amp;&amp;
CarRental.carType = "Toyota"
            </constraint>
            <execute template="CarRental"/>
            <ifthen name="Travel Choice">
                <if>Flight.cost &lt; Train.cost</if>
                <then>Flight.executionstatus == 1</then>
            </ifthen>
            <ifthen name="Travel Condition">
                <if>Flight.executionstatus == 1</if>
                <then>Flight.time == "27.04.07" &amp;&amp; Flight.from =="Ankara" &amp;&amp;
Flight.to == "Antalya"</then>
            </ifthen>
            <ifthen name="Travel Condition">
                <if>Train.executionstatus == 1</if>
                <then>Train.timeOfTravel &lt; 10</then>
            </ifthen>
            <ifthen name="Travel Condition">
                <if>Flight.cost &lt; 100</if>
                <then>Flight.executionstatus == 1</then>
            </ifthen>
            <ifthen name="Free Activity Condition">
                <if>Balloon.executionstatus == 1</if>
                <then>Balloon.duration&gt; 60</then>
            </ifthen>
            <ifthen name="Free Activity Condition 2">
                <if>Ski.executionstatus == 1</if>
                <then>Ski.time &gt; 1 &amp;&amp; Ski.instructor == "yes"</then>
            </ifthen>
            <control template1="Hotel" template2="CarRental" type="Same"/>
        </process-constraints>
    </composite-process>
</cwsl>
```

Below, constraint problem definition of above CWSL model is given;

```
Pb[59 vars, 54 cons]
Pb[59 vars, 54 cons]
==== VARIABLES ====
HOLIDAY.start:?[0, 2]
```

```
HOLIDAY.end:?[0, 2]
Model.start:?[0, 2]
Model.end:?[0, 2]
Model.executionstatus:?{0, 1}
Travel Required.start:?[0, 2]
Travel Required.end:?[0, 2]
Travel Required.executionstatus:?{0, 1}
Hotel.start:?[0, 2]
Hotel.end:?[0, 2]
Hotel.executionstatus:?{0, 1}
Travel Vehicle.start:?[0, 2]
Travel Vehicle.end:?[0, 2]
Travel Vehicle.executionstatus:?{0, 1}
Flight.start:?[0, 2]
Flight.end:?[0, 2]
Flight.executionstatus:?{0, 1}
Train.start:?[0, 2]
Train.end:?[0, 2]
Train.executionstatus:?{0, 1}
Free Activity.start:?[0, 2]
Free Activity.end:?[0, 2]
Free Activity.executionstatus:?{0, 1}
Balloon.start:?[0, 2]
Balloon.end:?[0, 2]
Balloon.executionstatus:?{0, 1}
Ski.start:?[0, 2]
Ski.end:?[0, 2]
Ski.executionstatus:?{0, 1}
CarRental.start:?[0, 2]
CarRental.end:?[0, 2]
CarRental.executionstatus:?{0, 1}
Hotel.cost:?{44, 46, 80, 83}
Hotel.distanceToCityCentre:?{6, 8, 11}
Hotel.location:?{0}
Hotel.star:?{2, 3, 6}
Hotel.numberOfDays:?{2, 4, 5}
Hotel.name:?{1}
Flight.cost:?{73, 88, 101, 120, 143, 164, 182, 197}
Flight.from:?{2}
Flight.to:?{0}
Flight.time:?{3}
Train.cost:?{19, 26}
Train.from:?{2}
Train.to:?{0}
Train.timeOfTravel:?{4, 5}
Balloon.cost:?{7, 8, 13, 14, 15, 17}
Balloon.duration:?{61, 77, 112, 123, 163, 164}
Ski.cost:?{28, 32, 44, 65}
```

Ski.time:?{4}
Ski.instructor:?{5}
CarRental.cost:?{57, 88}
CarRental.carType:?{6}
CarRental.motor:?{7}
temp.Flight.cost:?{0, 73, 88, 101, 120, 143, 164, 182, 197}
temp.Train.cost:?{0, 19, 26}
temp.Balloon.cost:?{0, 7, 8, 13, 14, 15, 17}
temp.Ski.cost:?{0, 28, 32, 44, 65}
Hilton Hotel.extra:?{0, 200}

==== CONSTRAINTS ====

HOLIDAY.start:? = HOLIDAY.end:? - 2
Model.start:? = Model.end:? - 1
Travel Required.start:? = Travel Required.end:? - 1
Travel Vehicle.start:? = Travel Vehicle.end:?
Free Activity.start:? = Free Activity.end:?
Travel Required.start:? >= Model.start:?
Model.end:? >= Travel Required.end:?
Free Activity.start:? >= Model.start:?
Model.end:? >= Free Activity.end:?
Hotel.start:? = Travel Required.start:?
Travel Vehicle.end:? = Travel Required.end:?
Travel Vehicle.start:? >= Hotel.end:?
(((Flight.start:? >= Travel Vehicle.start:?) and (Travel Vehicle.end:? >= Flight.end:?)) and (Flight.executionstatus:? == 1) and (Train.executionstatus:? == 0)) or (((Train.start:? >= Travel Vehicle.start:?) and (Travel Vehicle.end:? >= Train.end:?)) and (Flight.executionstatus:? == 0) and (Train.executionstatus:? == 1))
Balloon.start:? >= Free Activity.start:?
Free Activity.end:? >= Balloon.end:?
Ski.start:? >= Free Activity.start:?
Free Activity.end:? >= Ski.end:?
choco.integer.constraints.TimesXYZ@1ee9422
choco.integer.constraints.TimesXYZ@3f5cc3
choco.integer.constraints.TimesXYZ@1e33378
choco.integer.constraints.TimesXYZ@1481fbd
(Hotel.name:? == 1) => (Hilton Hotel.extra:? == 200)
-1*Hotel.cost:? + -1*temp.Flight.cost:? + -1*temp.Train.cost:? + -1*temp.Balloon.cost:? + -1*temp.Ski.cost:? + -1*CarRental.cost:? + -1*Hilton Hotel.extra:? >= -1499
Hotel.location:? == 0
Hotel.distanceToCityCentre:? <= 14
Hotel.star:? >= 4
(Hotel.numberOfDays:? != 4) and (Hotel.cost:? <= 599)
choco.integer.constraints.TimesXYZ@1bcc9f
choco.integer.constraints.TimesXYZ@116e402
-1*temp.Balloon.cost:? + -1*temp.Ski.cost:? >= -249
CarRental.executionstatus:? == 1

```
choco.integer.constraints.TimesXYZ@1d37013
choco.integer.constraints.TimesXYZ@631c70
( temp.Train.cost:? >= temp.Flight.cost:? + 1 ) => ( Flight.executionstatus:? == 1 )
( Flight.executionstatus:? == 1 ) => (  (Flight.time:? == 3) and (Flight.from:? ==
2) and (Flight.to:? == 0)  )
( Train.executionstatus:? == 1 ) => ( Train.timeOfTravel:? <= 9 )
choco.integer.constraints.TimesXYZ@e03bd6
( temp.Flight.cost:? <= 99 ) => ( Flight.executionstatus:? == 1 )
( Balloon.executionstatus:? == 1 ) => ( Balloon.duration:? >= 61 )
( Ski.executionstatus:? == 1 ) => (  (Ski.time:? >= 2) and (Ski.instructor:? == 5)
)
Model.start:? = HOLIDAY.start:?
CarRental.end:? = HOLIDAY.end:?
CarRental.start:? >= Model.end:?
Model.executionstatus:? == 1
Travel Required.executionstatus:? == 1
Hotel.executionstatus:? == 1
Travel Vehicle.executionstatus:? == 1
Free Activity.executionstatus:? == 1
 ( (Hotel.cost:? == 46) and (Hotel.distanceToCityCentre:? == 8) and
(Hotel.location:? == 0) and (Hotel.star:? == 6) and (Hotel.numberOfDays:? == 5) and
(Hotel.name:? == 1) ) or ( (Hotel.cost:? == 80) and (Hotel.distanceToCityCentre:? ==
11) and (Hotel.location:? == 0) and (Hotel.star:? == 3) and (Hotel.numberOfDays:? ==
4) and (Hotel.name:? == 1) ) or ( (Hotel.cost:? == 83) and
(Hotel.distanceToCityCentre:? == 6) and (Hotel.location:? == 0) and (Hotel.star:? ==
2) and (Hotel.numberOfDays:? == 4) and (Hotel.name:? == 1) ) or ( (Hotel.cost:? ==
44) and (Hotel.distanceToCityCentre:? == 8) and (Hotel.location:? == 0) and
(Hotel.star:? == 6) and (Hotel.numberOfDays:? == 2) and (Hotel.name:? == 1) )
 ( (Flight.cost:? == 101) and (Flight.from:? == 2) and (Flight.to:? == 0) and
(Flight.time:? == 3) ) or ( (Flight.cost:? == 182) and (Flight.from:? == 2) and
(Flight.to:? == 0) and (Flight.time:? == 3) ) or ( (Flight.cost:? == 73) and
(Flight.from:? == 2) and (Flight.to:? == 0) and (Flight.time:? == 3) ) or (
(Flight.cost:? == 197) and (Flight.from:? == 2) and (Flight.to:? == 0) and
(Flight.time:? == 3) ) or ( (Flight.cost:? == 88) and (Flight.from:? == 2) and
(Flight.to:? == 0) and (Flight.time:? == 3) ) or ( (Flight.cost:? == 143) and
(Flight.from:? == 2) and (Flight.to:? == 0) and (Flight.time:? == 3) ) or (
(Flight.cost:? == 120) and (Flight.from:? == 2) and (Flight.to:? == 0) and
(Flight.time:? == 3) ) or ( (Flight.cost:? == 164) and (Flight.from:? == 2) and
(Flight.to:? == 0) and (Flight.time:? == 3) )
 ( (Train.cost:? == 26) and (Train.from:? == 2) and (Train.to:? == 0) and
(Train.timeOfTravel:? == 5) ) or ( (Train.cost:? == 19) and (Train.from:? == 2) and
(Train.to:? == 0) and (Train.timeOfTravel:? == 4) )
 ( (Balloon.cost:? == 15) and (Balloon.duration:? == 77) ) or ( (Balloon.cost:? ==
8) and (Balloon.duration:? == 112) ) or ( (Balloon.cost:? == 7) and
(Balloon.duration:? == 61) ) or ( (Balloon.cost:? == 14) and (Balloon.duration:? ==
163) ) or ( (Balloon.cost:? == 17) and (Balloon.duration:? == 123) ) or (
(Balloon.cost:? == 13) and (Balloon.duration:? == 164) )
```

```
 ( (Ski.cost:? == 65) and (Ski.time:? == 4) and (Ski.instructor:? == 5) ) or (
(Ski.cost:? == 32) and (Ski.time:? == 4) and (Ski.instructor:? == 5) ) or (
(Ski.cost:? == 44) and (Ski.time:? == 4) and (Ski.instructor:? == 5) ) or (
(Ski.cost:? == 28) and (Ski.time:? == 4) and (Ski.instructor:? == 5) )
 ( (CarRental.cost:? == 57) and (CarRental.carType:? == 6) and (CarRental.motor:? ==
7) ) or ( (CarRental.cost:? == 88) and (CarRental.carType:? == 6) and
(CarRental.motor:? == 7) )
```

In the following part, constraint engine solution of the above problem is given;

```
HOLIDAY.start:0 = 0
HOLIDAY.end:2 = 2
Model.start:0 = 0
Model.end:1 = 1
Model.executionstatus:1 = 1
Travel Required.start:0 = 0
Travel Required.end:1 = 1
Travel Required.executionstatus:1 = 1
Hotel.start:0 = 0
Hotel.end:0 = 0
Hotel.executionstatus:1 = 1
Travel Vehicle.start:1 = 1
Travel Vehicle.end:1 = 1
Travel Vehicle.executionstatus:1 = 1
Flight.start:1 = 1
Flight.end:0 = 0
Flight.executionstatus:1 = 1
Train.start:0 = 0
Train.end:0 = 0
Train.executionstatus:0 = 0
Free Activity.start:0 = 0
Free Activity.end:0 = 0
Free Activity.executionstatus:1 = 1
Balloon.start:0 = 0
Balloon.end:0 = 0
Balloon.executionstatus:1 = 1
Ski.start:0 = 0
Ski.end:0 = 0
Ski.executionstatus:1 = 1
CarRental.start:1 = 1
CarRental.end:2 = 2
CarRental.executionstatus:1 = 1
Hotel.cost:44 = 44
Hotel.distanceToCityCentre:8 = 8
Hotel.location:0 = 0
Hotel.star:6 = 6
Hotel.numberOfDays:2 = 2
Hotel.name:1 = 1
Flight.cost:73 = 73
Flight.from:2 = 2
Flight.to:0 = 0
Flight.time:3 = 3
Train.cost:19 = 19
Train.from:2 = 2
Train.to:0 = 0
Train.timeOfTravel:4 = 4
Balloon.cost:7 = 7
Balloon.duration:61 = 61
```

```
Ski.cost:65 = 65
Ski.time:4 = 4
Ski.instructor:5 = 5
CarRental.cost:88 = 88
CarRental.carType:6 = 6
CarRental.motor:7 = 7
temp.Flight.cost:73 = 73
temp.Train.cost:0 = 0
temp.Balloon.cost:7 = 7
temp.Ski.cost:65 = 65
Hilton Hotel.extra:200 = 200
```

A.2 Traveling Domain Example 2

This example focuses on concepts of CWSL that are not shown in the section A.1. Usage of query Constraints, process variables, transition expressions and decision and iteration blocks are given especially in this example. Example displays different model in travel domain, which is given in Figure A.1. In this model, there exist few numbers of constraints.

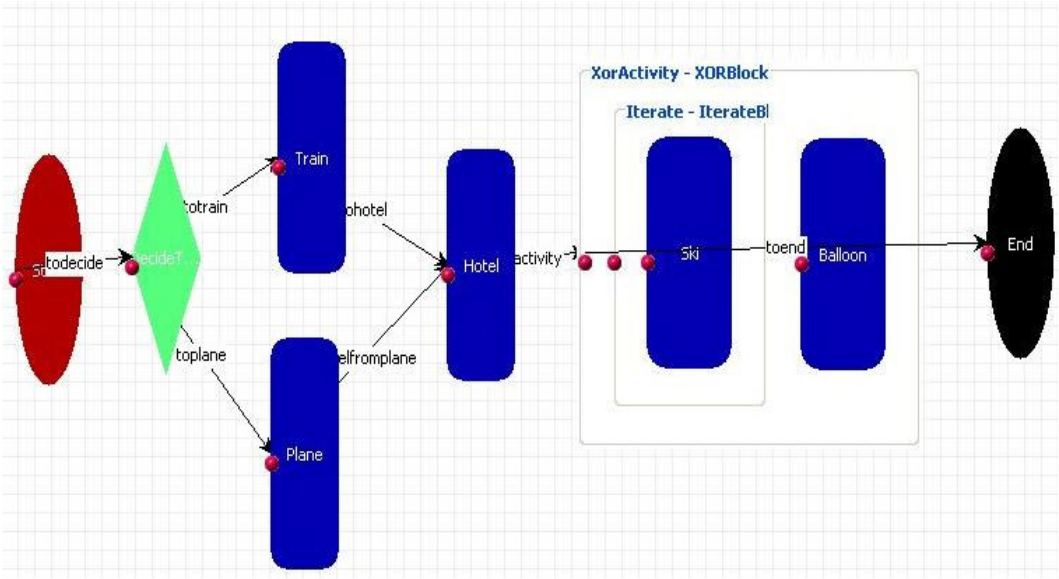

Figure A.2 Travel Planner Example 2

Full CWSL model of the model is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<cwsl name="Example2">
```

```
<composite-process>
 <ontology>
  <url>TravelingOntology</url>
 </ontology>
 <sequence name="Example2">
  <start name="Start">
   <transition name="todecide" to="DecideTravelMethod"/>
  </start>
  <decision name="DecideTravelMethod">
   <transition name="totrain" to="Train">
   if(travelChoice.equals("Train"))
   return true;
   </transition>
   <transition name="toplane" to="Plane">
    if(travelChoice.equals("Plane"))
    return true;
  </transition>
  </decision>
  <service-template name="Train">
   <ontology-class>
    <owlClassModel>Train</owlClassModel>
    <action>resyen</action>
   </ontology-class>
   <variable-set>
    <variable assign-method="minimize" name="cost" type="Int"/>
   </variable-set>
   <constraint-set/>
   <query>
    <criteria>
     <constraint attribute="from" op="equal" value="Ankara"/>
     <constraint attribute="to" op="equal" value="Antalya"/>
    </criteria>
   </query>
   <transition name="tohotel" to="Hotel"/>
  </service-template>
  <service-template name="Plane">
   <ontology-class>
    <owlClassModel>Plane</owlClassModel>
    <action>resplane</action>
   </ontology-class>
   <variable-set>
    <variable assign-method="minimize" name="cost" type="Int"/>
   </variable-set>
   <constraint-set/>
   <query>
    <criteria>
     <constraint attribute="from" op="equal" value="Ankara"/>
     <constraint attribute="to" op="equal" value="Ankara"/>
     <constraint attribute="duration" op="lessthan" value="120"/>
    </criteria>
   </query>
   <transition name="tohotelfromplane" to="Hotel"/>
  </service-template>
  <service-template name="Hotel">
   <ontology-class>
    <owlClassModel>Hotel</owlClassModel>
    <action>reshotel</action>
   </ontology-class>
   <variable-set>
    <variable assign-method="minimize" name="cost" type="Int"/>
   </variable-set>
   <constraint-set/>
   <query>
    <criteria>
     <constraint attribute="location" op="equal" value="Ankara"/>
    </criteria>
   </query>
```

```
      <transition name="tofreeactivity" to="XorActivity"/>
    </service-template>
    <end name="End"/>
    <xor name="XorActivity">
      <service-template name="Balloon">
        <ontology-class>
          <owlClassModel>Balloon</owlClassModel>
          <action>resballoon</action>
        </ontology-class>
        <variable-set>
          <variable assign-method="minimize" name="cost" type="Int"/>
        </variable-set>
        <constraint-set/>
        <query>
          <criteria/>
        </query>
      </service-template>
      <iterate assign-method="maximize" name="Iterate" variable="skiiterate">
        <service-template name="Ski">
          <ontology-class>
            <owlClassModel>Ski</owlClassModel>
            <action>ressk</action>
          </ontology-class>
          <variable-set>
            <variable assign-method="minimize" name="cost" type="Int"/>
          </variable-set>
          <constraint-set/>
          <query>
            <criteria/>
          </query>
        </service-template>
      </iterate>
      <transition name="toend" to="End"/>
    </xor>
  </sequence>
  <business-rules/>
  <process-variable-set>
    <variable name="travelChoice" type="String" value="Train"/>
  </process-variable-set>
  <expression-set/>
  <process-constraints>
    <constraint name="some">
Hotel.cost &lt; 500
</constraint>
    <execute template="Ski"/>
    <ifthen name="Balloon Rule">
      <if>Balloon.executionstatus == 1</if>
      <then>Balloon.cost &lt; 110</then>
    </ifthen>
    <ifthen name="Ski rule">
      <if>Ski.executionstatus == 1</if>
      <then>(skiiterate * 2) &lt; 10</then>
    </ifthen>
  </process-constraints>
 </composite-process>
</cwsl>
```

Below, constraint problem definition of above CWSL model is given;

```
Pb[32 vars, 21 cons]
Pb[32 vars, 21 cons]
==== VARIABLES ====
Example2.start:?[0, 3]
```

```
Example2.end:?[0, 3]
DecideTravelMethod.start:?[0, 3]
DecideTravelMethod.end:?[0, 3]
DecideTravelMethod.executionstatus:?{0, 1}
Train.start:?[0, 3]
Train.end:?[0, 3]
Train.executionstatus:?{0, 1}
Plane.start:?[0, 3]
Plane.end:?[0, 3]
Plane.executionstatus:?{0, 1}
Hotel.start:?[0, 3]
Hotel.end:?[0, 3]
Hotel.executionstatus:?{0, 1}
XorActivity.start:?[0, 3]
XorActivity.end:?[0, 3]
XorActivity.executionstatus:?{0, 1}
Balloon.start:?[0, 3]
Balloon.end:?[0, 3]
Balloon.executionstatus:?{0, 1}
Iterate.start:?[0, 3]
Iterate.end:?[0, 3]
Iterate.executionstatus:?{0, 1}
Ski.start:?[0, 3]
Ski.end:?[0, 3]
Ski.executionstatus:?{0, 1}
skiiterate:?{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14..., 100}
Train.cost:30{30}
Hotel.cost:?{75, 77}
Balloon.cost:?{8, 19, 20}
Ski.cost:?{60, 64}
temp.Balloon.cost:?{0, 8, 19, 20}
Pb[32 vars, 21 cons]
```

==== CONSTRAINTS ====

```
Example2.start:? = Example2.end:? - 3
XorActivity.start:? = XorActivity.end:?
Iterate.start:? = Iterate.end:?
 ( ( (Balloon.start:? >= XorActivity.start:?) and (XorActivity.end:? >=
Balloon.end:?) ) and (Balloon.executionstatus:? == 1) and (Iterate.executionstatus:?
== 0) ) or ( ( (Iterate.start:? >= XorActivity.start:?) and (XorActivity.end:? >=
Iterate.end:?) ) and (Balloon.executionstatus:? == 0) and (Iterate.executionstatus:?
== 1) )
Ski.start:? >= Iterate.start:?
Iterate.end:? >= Ski.end:?
Hotel.cost:? <= 499
Ski.executionstatus:? == 1
choco.integer.constraints.TimesXYZ@ef8e87
( Balloon.executionstatus:? == 1 ) => ( temp.Balloon.cost:? <= 109 )
( Ski.executionstatus:? == 1 ) => ( -2*skiiterate:? >= -9 )
```

```
DecideTravelMethod.start:? = Example2.start:?
XorActivity.end:? = Example2.end:?
Train.start:? >= DecideTravelMethod.end:?
Plane.start:? >= Train.end:?
Hotel.start:? >= Plane.end:?
XorActivity.start:? >= Hotel.end:?
Train.executionstatus:? == 1
Plane.executionstatus:? == 1
Hotel.executionstatus:? == 1
XorActivity.executionstatus:? == 1
```

In the following part, constraint engine solution of the above problem is given;

```
Example2.start:0 = 0
Example2.end:3 = 3
DecideTravelMethod.start:0 = 0
DecideTravelMethod.end:0 = 0
DecideTravelMethod.executionstatus:1 = 1
Train.start:0 = 0
Train.end:0 = 0
Train.executionstatus:1 = 1
Plane.start:0 = 0
Plane.end:0 = 0
Plane.executionstatus:1 = 1
Hotel.start:0 = 0
Hotel.end:0 = 0
Hotel.executionstatus:1 = 1
XorActivity.start:3 = 3
XorActivity.end:3 = 3
XorActivity.executionstatus:1 = 1
Balloon.start:3 = 3
Balloon.end:0 = 0
Balloon.executionstatus:1 = 1
Iterate.start:0 = 0
Iterate.end:0 = 0
Iterate.executionstatus:0 = 0
Ski.start:0 = 0
Ski.end:0 = 0
Ski.executionstatus:1 = 1
skiiterate:4 = 4
Train.cost:30 = 30
Hotel.cost:75 = 75
Balloon.cost:8 = 8
Ski.cost:60 = 60
temp.Balloon.cost:8 = 8
```