An Intelligent Web Service Workflow: A Petri Net Based Approach

E. Hossny, S. AbdElrahman and A. Badr*

Department of Computer Science, Faculty of Computers and Information, Cairo University, Egypt

Abstract: Fuzzy Petri Net for Web Service Composition (FPN4WSC) aims to compose the individual web services into more complex one. It is a workflow model which is hybridization between Petri net, SHOP2, and fuzzy logic. Petri net allows user to specify his request as a workflow. SHOP2 is used as an Artificial Intelligence (AI) planning system to get a plan for the user request. However, SHOP2 fails to capture the uncertainty. So, the fuzzy logic is used as a refinement engine to get the best solution based on the user preferences. Therefore, FPN4WSC presents simple graphical, intelligent, and automatic web service composition model. FPN4WSC model is scalable to any workflow-based domains. As a case study, it is applied on the travel reservation domain where the user should specify his preferences and the fuzzy engine tries to get the best solution depending on his preferences.

Keywords: Web Services, Fuzzy Petri Net, SHOP2, Web Service Composition.

INTRODUCTION

Nowadays, the internet has become a service oriented model instead of a repository of information, where many companies are putting their core business on the web as a collection of web services [1-3]. Web service [1, 4] is considered as self-contained, self-describing, modular application that can be described, advertised, discovered, and executed through the web. It can be identified by URL, whose interfaces and bindings are described by using XML.

The emergency of web services has led to more interest for composing individual web services into more complex one. The ability to efficiently and effectively assemble the autonomous and heterogeneous web services on the internet at runtime is a critical step towards the development of web service applications. Actually, if there is no web service that can fulfill the user request, there should have a possibility to integrate set of existing web services together to satisfy the functionality required by the user [5]. Thus, web service composition has been an active area of research in the past few years. The study of workflow model is one of the most important parts of web service composition. Several approaches are proposed to investigate and model the process of web service composition. However, these approaches did not provide formal framework for modeling the automated and complex service composition process. Also, the current technologies, such as Universal Description, Discovery, and Integration (UDDI) [6], Web Service Description Language (WSDL) [7], and Simple Object Access Protocol (SOAP) [8], do not realize complex web service integrations. So, they give limited supported in web service composition [1]. The proposed system is a fuzzy Petri net based model for composing web services. This model is called Fuzzy Petri net for Web Service Composition (FPN4WSC). The web services are modeled as Petri nets by assigning transitions to services and places to states. The model uses SHOP2 as an AI planner to get a plan for the user request. Since SHOP2 can not capture the uncertainty, the fuzzy logic is used as a refinement engine to get the best solution based on the user preferences.

*Address correspondence to this author at the Department of Computer Science, Faculty of Computers and Information, Cairo University, Egypt; E-mail: s.abdelrahman@fci-cu.edu.eg

BACKGROUND

Petri Net

A Petri Net (PN) [9] is a directed, connected and bipartite graph. It was invented in 1962 by Carl Adam Petri. A Petri net has four components, namely place nodes, transition nodes, directed arcs connecting places with transitions, and tokens occupy places. Petri nets have a well defined mathematical foundations and an easy-to-understand graphical feature. The graphical nature of Petri nets makes them self-documenting and a powerful design tool which facilitate the visual communication between the members who are engaged in the design [10].

Definition 1 (Petri Net)

A Petri net is an algebraic structure (P, T, F, M0):

1. **P** is a finite set of places**,** **T** is a finite set of transitions and **P** ∩ **T** = Φ
2. **F** $\subseteq (PxT) \cup (TxP)$ is a set of directed arcs from **P** to **T** and from **T** to **P**
3. **M0** is the initial marking.

SHOP2

SHOP2 [11] is a domain-independent Hierarchical Task Network (HTN) planning system. HTN planning is an AI planning methodology that creates plan by task decomposition. The task decomposition is a process in which the planning system decomposes tasks into smaller and smaller subtasks, until primitive tasks are found that can be performed directly. This makes HTN planning system a good candidate for automatic Web services composition task.

One difference between SHOP2 and other HTN planning systems is that SHOP2 plans for tasks in the same order that they will later be executed. Planning for tasks in the order that those tasks will be performed makes it possible to know the current state of the world at each step in the planning process. While SHOP2 allows for tasks to be sequentially ordered, there is no mechanism to handle the control constructs related to concurrency, namely parallel split, synchronization, and exclusive choice. The proposed model overcame this problem by adding a new keyword (:concurrent) to handle the concurrency problem.

In order to do planning in a given domain, SHOP2 needs to be given knowledge about that domain. The knowledge based of SHOP2 contains set of operators and methods. Each operator is a description of what needs to be done to execute some primitive task. Each method tells how to decompose some compound task into partially ordered subtasks.

There exists a java implementation for SHOP2 called JSHOP2 [12] which is an open source. The inputs to JSHOP2 are a planning domain and a planning problem. The planning domain is composed of operators and methods. The planning problem is composed of an initial state and list f tasks to be performed. The planning domain and problem are written by the lisp language [13].

RELATED RESEARCH

Several approaches investigated the process of web service composition. However, none of these approaches offers formal framework for modeling the automated and complex service composition process. The proposed model addresses a particular subset of this problem with FPN4WSC, which is a fuzzy Petri-net-based workflow model for web service composition. In this section, the approaches that are closely related to our work are briefly described.

Web service composition requires more complex functionalities such as transactions, workflow, negotiation, management, security. Those functionalities are not provided by the current technologies based on WSDL, SOAP, and UDDI. There are several efforts that aim at providing such functionalities, for example, the Business Process Execution Language for Web Services (BPEL4WS) [14] is positioned to become the basis of a standard for web service composition. But this language is complex procedural language and very hard to implement and deploy [2]. Its XML representation is very verbose and it has many constructs. Also, it facilitates orchestration only, i.e. allows execution of a manually constructed composition. There are also some proposals such as OWL-S [15] that aims at realizing the semantic web concept. However OWL-S is a complex procedural language for web service composition.

In the workflow community, a lot of attention has been paid to allow the systems to be adaptive and to separate between the interface and implementation of a process. eFlow [16] is one of the recent emerging workflow projects focus on loosely coupled processes. It supports adaptive and dynamic service composition. However, it lacks a formal model to specify and verify the web service composition [2].

Petri nets have been widely used as a tool for workflow modeling and analysis since their introduction by C.A. Petri in 1962 . Hamadi [2] proposed a Petri-net-based algebra to model control flows, as a necessary constituent of reliable web service composition process. The model can capture the semantics of complex web service composition and enable declarative composition of web services. Also it aims to provide support for correct web services by insuring the absence of deadlocks and live locks. However, it lacks a management of time and resources. Service/Resource Net (SRN) [1] is an extended Petri-net-based model for web service composition with some new elements such as time, resource taxonomy, and condition. It proposed Web Service Semigroup (WSSG) and meta-service based on group theory as a theoretical system of SRN service taxonomy. This model is more closely related to the model of Hamadi [2]. Bing and Huaping [10] proposed a Petri-net-based algebra to capture the semantics of complex web service composition. In this model the web services are represented as Petri nets where transitions are assigned to methods and places are assigned to states. The model uses the Petri net to test a set of non-functional properties such as reachability, safety and existence of deadlocks. However, it did not provide a formal framework for modeling the automated and complex web service composition. Evren *et al.* [11] integrated the SHOP2 planning system with DAML-S web services descriptions to automatically compose the web services. The authors used the DAML-S for semantic markup of web services and gave a detailed description of how to translate DAML-S process definitions into a SHOP2 domain. Also, they implemented a converter to convert a SHOP2 plan to DAML-S format which can be executed directly by a DAML-S executor. However, that approach can not handle the control constructs related to concurrency.

Fu *et al.* [17] proposed an optimized web service composition algorithm based on fuzzy Petri net and semantic web. In this algorithm the web services are described by fuzzy Horn clause and the Quality of Service (QoS)-oriented web service composition model is built by fuzzy Petri Net.

FPN4WSC MODEL

The process of web service composition can be regarded as workflow. Workflow model is the precondition of workflow. The Petri net can be used as a practical method and tool to model the workflows. The basic definition and concept of Petri net can be found in [9]. The relation between Petri net and workflow net is defined as the following [1]:

Definition 2

A Petri net is a workflow net iff:

1. has two specific places , source place (*i*) and sink place (*o*) $\bullet i = \phi, o\bullet = \phi$
2. if a new transition (*t*) is added to connect place *o* and place *i*, i.e. $\bullet t = \{o\}, t\bullet = \{i\}$, PN is strong connected Petri net

To describe the workflow of web service composition, a new model is proposed as hybridization between Petri net, SHOP2, and fuzzy logic. Petri net helps the user to draw his request as a workflow. SHOP2 is used as an AI planning system to get a plan for the user request. Since SHOP2 fails to capture the uncertainty, the fuzzy logic is used as a refinement engine to get the best solution based on the user preferences. This model is called Fuzzy Petri net for Web Service Composition (FPN4WSC).

Most precisely, the process of automatic service composition includes the following phases [5].

Presentation of Service

The service providers publish their atomic services at a global service registry, such as, UDDI. The essential attributes to describe a web service include the service's inputs, outputs and exceptions. In the proposed model, the services are represented using WSDL.

Translation of the Languages

Most service composition systems distinguish between the external and internal service specification languages. The external languages are used by the service requesters to allow them to express what they want in a relatively easy manner. They are usually different from the internal ones that are used by the composition process generator, because the process generator requires more formal and precise languages, for example, the logical programming languages and SHOP2 [18].

Generation of Composition Process Model

The service requester can also express the requirement in a service specification language. Then, a process generator tries to solve the requirement by composing the atomic services which are advertised by the service providers. The process generator usually takes the functionalities of services as input, and outputs a plan that describes the composite service.

Evaluation of Composite Service

It is quite common that many services have the same or similar functionalities. So, the planner might generate more than one composite service fulfilling the requirement. In that case, the composite services are evaluated by their overall utilities using the information provided from the non-functional attributes. The most commonly used method is utility functions. The requester should specify weights to each non-functionality attributes and the best composite service is the one who is ranked on top.

Process Execution Engine

After a unique composite process is selected, the composite service is ready to be executed. Execution of a composite web service can be thought as a sequence of message passing according to the plan. The dataflow of the composite service is defined as the actions that the output data of a former executed service transfers to the input of a later executed atomic service.

RESEARCH CONTRIBUTION

A hybrid system is innovated to achieve simple graphical, intelligent, and automatic web service composition that provides the following main contributions:

- The user preferences are satisfied using uncertainty values for each preference through automatic generation of suitable plans.
- The proposed system is scalable to any workflows-based domains. This means the user can get the WSDL description for a set of web services and add them to the system without any problem.
- The difficulties of each individual component are solved. For example, Petri net can only generate workflows and SHOP2 can only generate plans. These two components are combined with the fuzzy to generate an intelligent graphical automatic web service composition model.

SYSTEM DESIGN

FPN4WSC can be modeled as a client server model, where the client asks the server to execute a set of web services represented as a Petri net workflow and the server tries to satisfy his request and provide him with the best solution.

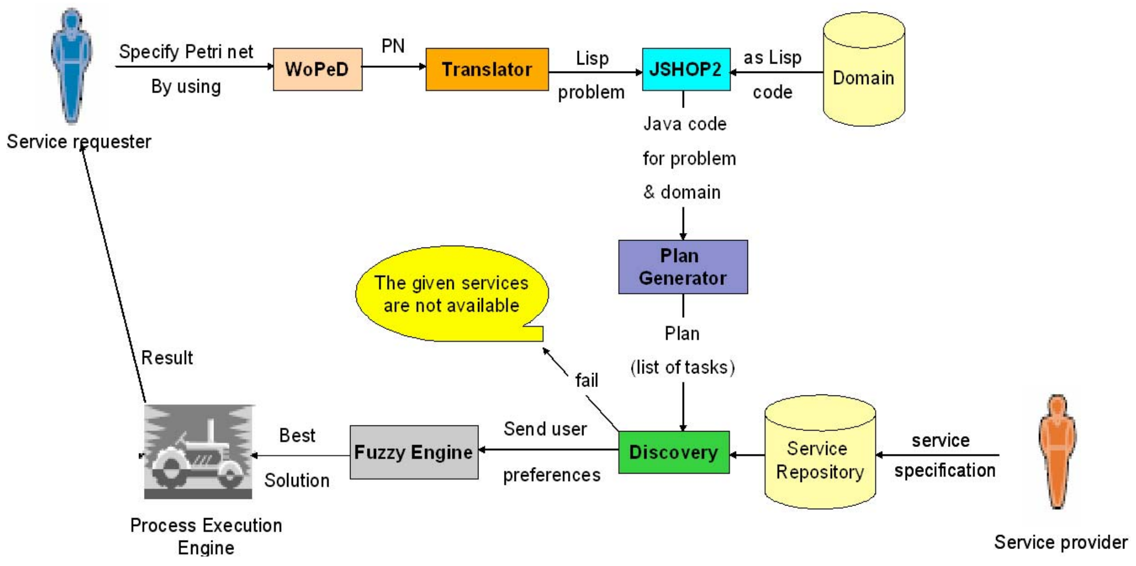


Fig. (1). Fuzzy Petri net model for web service composition.

The architecture of the proposed model is depicted in Fig. (**1**). The composition system of the proposed model has two types of participants, namely service provider and service requester. The service providers publish their services for use. The service requesters use the services offered by the service providers. The composition system also contains the following components: WoPeD (Workflow Petri net Designer), translator, JSHOP2, plan generator, service repository, discovery engine, evaluator (fuzzy engine), and execution engine. A full description of each component is illustrated in the following subsections. Algorithm 1 specifies how all these components integrated with each other.

Algorithm 1 System Components Integration

1. The service requester should specify his request (i.e. a set of web services to be composed) using WoPeD.
2. WoPeD generates a Petri net workflow that is represented the user request.
3. The translator component (Algorithm 2) is built to parse the output from WoPeD (i.e. a Petri net workflow) into lisp problem which is entered as an input to SHOP2.
4. SHOP2 executes the lisp problem and generates a plan that is represented as a java code.
5. The plan generator component (Algorithm 3) is built to execute the java code of the plan and generate a plan which is a list of tasks (web services) to be executed.
6. An input form is displayed to the service requester to ask about set of required data and the user preferences.
7. The discovery component (Algorithm 4) is built to detect if the required web services and their inputs are available.
8. If the discovery component fails in the previous step, then a message is displayed to the user to tell him that the required services are not available.
9. Otherwise, the discovery component sends the user preferences to the fuzzy engine.
10. The fuzzy engine is used as a refinement engine to get the best solution based on set of non-functional attributes such as uncertainty constraints. The non-functional attributes are represented through the user preferences.
11. WSDL2OWLS [19] converter is used to convert WSDL description of web service into its equivalent OWL-S.
12. Finally, the process execution engine (Algorithm 5) executes the OWL-S of the required web services according to the plan.

A full description of the proposed system components is illustrated in the following subsections.

WoPeD (Workflow Petri Net Designer)

WoPeD [20] is an open source java code for designing Petri net workflows. It can be used by the user to enhance accessibility of the users in the sense that the users can express what they want in a relatively easy manner using the drag/drop of the elements in the Graphical User Interface (GUI) of WoPeD. It has a friendly user interface which can be used by the end user in an easy manner. The user can use this open source to specify his request as a set of places and transitions. The places are used to represent the states (i.e. jump from a web service into another). The transitions are used to represent the web services. For example, use WoPeD to draw two concurrently web services, namely SearchFlight and SearchHotel, and one sequential service called ReserveHotel. The Petri net for this example is shown in Fig. (**2**).

JSHOP2

As it is mentioned above, JSHOP2 [12] is an open source java code which plan for the tasks to be ordered sequentially and there is no mechanism to handle the control constructs related to concurrency, so the proposed model overcame this problem by adding a new keyword (:concurrent) to JSHOP2 in order to handle the concurrency problem and allow tasks to be executed concurrently. The inputs to JSHOP2 are planning domain and problem files. Those files must be written by lisp language.

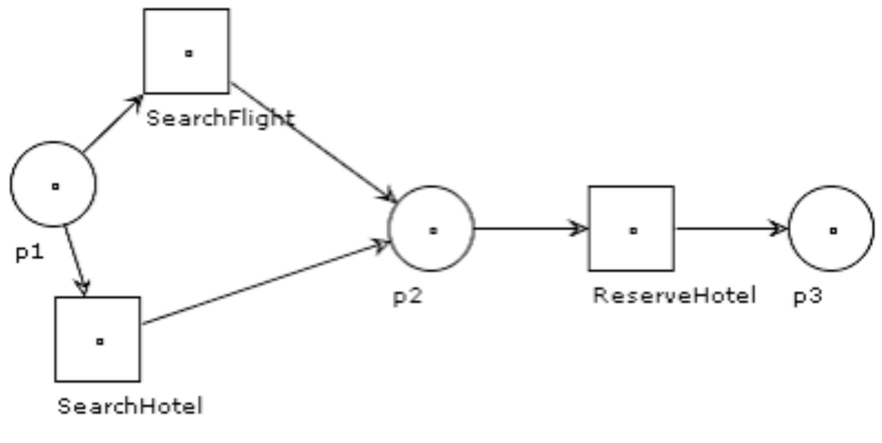


Fig. (2). Petri net example is designed by WoPeD.

A sample of the planning domain whose Petri net is clarified in Fig. (**2**), is depicted as the following:

Lisp code for the planning domain:

```
(defdomain webservices
 (   (:operator (!searchFlight)
             ((from ?x)(to ?y)) ()
             ((flight ?x ?y))
     )
     (:method (SearchFlight)
             ((from ?x)(to ?y))
             ((!searchFlight))
     )
     (:operator (!reserveflight)
             ((from ?x)(to ?y)) ()
             ((flight ?x ?y reserved))
     )
     (:method (ReserveFlight)
             ((from ?x)(to ?y))
             ((!reserveflight))
     )
     (:operator (!searchHotel)
             ((destination ?x)) ()
             ((hotel ?x))
     )
     (:method (SearchHotel)
             ((destination ?x))
             ((!searchHotel))
     )
))
```

Translation of the Languages

The language used by the JSHOP2 is different from the language used to represent Petri net in the WoPeD. WoPeD generates a Petri net workflow. The JSHOP2 needs the problem to be represented by the lisp language. Thus the translation component between the WoPeD language and the internal language (i.e. lisp) is developed. Algorithm 2 specifies the functionalities of the translator component.

Algorithm 2 Translator Component

1. Travers the paths of the Petri net network and save them as a vector, where each element in the vector contains a path and its type (sequential or concurrent).
2. Translate the paths vector into lisp code according to the syntax of the lisp language, such that the sequential task is written as it is between () and the concurrent tasks are preceded by the keyword :concurrent. For example, the generated lisp code for the Petri net in Fig. (**2**) is clarified in Fig. (**3**).
3. Finally, the generated lisp code represents the problem and enters as an input to SHOP2.

Generation of Composition Process Model

The JSHOP2 outputs the java code which represents the problem and domain. This java code is entered as an input to the plan generator component to generate suitable plans for the composed web services. Algorithm 3 clarifies the functionalities of the plan generator component.

Algorithm 3 Plan generator Component

1. Compiles the java code of the problem and domain. This compilation generates executable code for the problem.
2. Run the executable code of the problem and save the plan into a text file.
3. Parse the plan text file and determine the concurrent and the sequential tasks.
4. Finally, save the parsed data as a vector, where each element in the vector represents a web service name.

Discovery

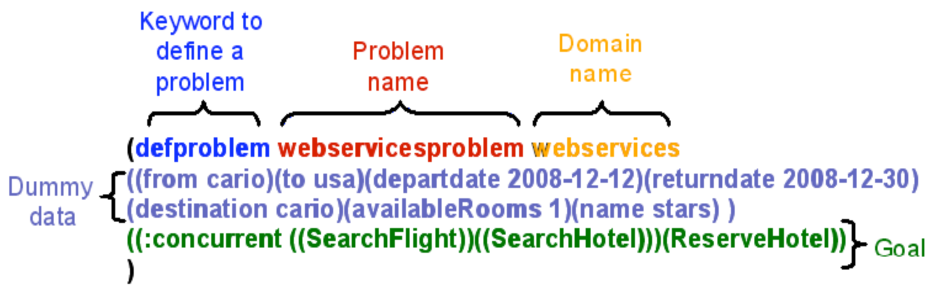
There exists a database which stores the available web services in the current domain. The discovery component is developed to detect if the required web services and their inputs are available. Algorithm 4 depicts how the discovery component works.

Algorithm 4 Discovery Component

1. Create SQL statement to search in the database about each web service with the given data.
2. If the previous step fails, then a message is displayed to the user to tell him that the required services are not available.
3. Otherwise, the discovery component sends the user preferences to the fuzzy engine to get the best solution for this user.

Evaluation of Composite Service

This can be satisfied using fuzzy engine. The fuzzy engine contains set of fuzzy rules and linguistic variables. In the proposed model the fuzzy rules are static and must be defined based on the given domain.



```
(defproblem webservicesproblem webservices
((from cario)(to usa)(departdate 2008-12-12)(returndate 2008-12-30)
(destination cario)(availableRooms 1)(name stars) )
((:concurrent ((SearchFlight))((SearchHotel)))(ReserveHotel))
)
```

Fig. (3). Lisp code sample that is generated from the translator component.

All fuzzy evaluations are based on the rules in the symbolic representation. The following is the format for symbolic representation of fuzzy rule [21]:

```
if LV1 is MF1 <and/or> LV2 is MF2 …
```

then LVN is MFN

This format can be used to define the static fuzzy rules and then load them to the fuzzy engine. The user provides the system with a set of his preferences and the fuzzy engine runs the fuzzy rules to get the best solution depending on the user preferences. Since there is a set of uncertainty values in a web service, the user should specify at least two preferences and specify the status of each preference, rigid or soft. After the preferences are specified, the fuzzy engine runs and tries to satisfy the first preference. Then runs again to satisfy the second preference and finally makes a filtration based on status of the first and the second preferences and get the best solution. Where, if there is contradiction between the first and the second preferences, then the fuzzy engine tries to execute the rigid one. For example, first preference: weight is large and status is rigid, second preference: budget is large and status is soft. Since the weight is large, this means the user can not travel by plan. Since the budget is large, the user can take the first class of a vehicle. Thus the final output from the fuzzy engine is "You can take bus and the class of service is first"

Process Execution Engine

The plan of composed services runs by the process execution engine. The required services must be described by OWL-S and enters as inputs to the process execution engine. Algorithm 5 specifies how the process execution engine works.

Algorithm 5 Process Execution Engine

1. Travers the plan vector and get the web service name in each element in the vector.
2. Load the OWL-S description of the current web service name, such that each concurrent web service runs in a separate thread and the sequential web services run sequentially in the same thread.

IMPLEMENTATION

Actually, the proposed model is mainly implemented using the java language and Eclipse Integrated Development Environment (IDE). Since the java language is a platform independent, the proposed system can be used in any platform. The end user can run the proposed system in two ways, standalone java application or java applet. But in both cases he must specify his request as a Petri net workflow using the simple graphical user interface of WoPeD.

The main three components which are used in the proposed system, namely WoPeD, fuzzy engine, and JSHOP2, are open sources java code and some of modifications are added to enhance each one of them. They can be downloaded from [20-22]. Below are the modifications which are added to those open sources:

JSHOP2

Since JSHOP2 has no mechanism to handle the control constructs related to concurrency, the proposed model overcame this problem by adding a new keyword (:concurrent) to JSHOP2 in order to handle the concurrency problem and allow tasks to be executed concurrently.

Fuzzy engine

Depending on the domain which the model is applied on, a new class is added to define the required linguistic variables and the fuzzy rules. In the proposed model, the fuzzy rules are static and depending on the domain. In the case study, the proposed model is applied on the travel reservation system and a set of the web services required in this domain are implemented. Below is a set of fuzzy rules which are added to the system for travel reservation domain:

"if maxWeight is small then flight is firstclass",

"if maxWeight is medium then bus is firstclass",

"if maxWeight is large then surface is firstclass",

"if classOfService is first then budget is firstclass",

"if classOfService is business then budget is secondclass",

"if classOfService is economy then budget is thirdclass",

"if tripSpeed is fast then budget is firstclass",

"if tripSpeed is medium then budget is secondclass",

"if tripSpeed is slow then budget is thirdclass",

"if budget is firstclass then flight is firstclass",

"if budget is secondclass then flight is businessclass and bus is firstclass",

"if budget is thirdclass then flight is economyclass and bus is businessclass and surface is firstclass",

"if hotelCOS is standard then pricePerDay is same",

"if hotelCOS is superior then pricePerDay is increased",

"if roomType is singleRoom then pricePerDay is same",

"if roomType is doubleRoom then pricePerDay is increased",

WoPeD

WoPeD is a tool for designing Petri net workflows and the user uses it to draw his workflow. Some code is added to WoPeD to parse the user workflow and get the concurrent and the sequential paths which exist in the workflow. Note that the JSHOP2 and the fuzzy engine are used inside the WoPeD. Therefore, WoPeD becomes the main controller which runs the overall system. Also, another three components are built, namely translator, plan generator, and discovery engine, inside the WoPeD. The translator component is used to translate the workflow designed by WoPeD into lisp code. This lisp code enters to JSHOP2 as an input and then JSHOP2 outputs a java code. The plan generator component uses this java code as an input and generates plans for the user request. The discovery engine detects if the required request can be satisfied based on the available web services.

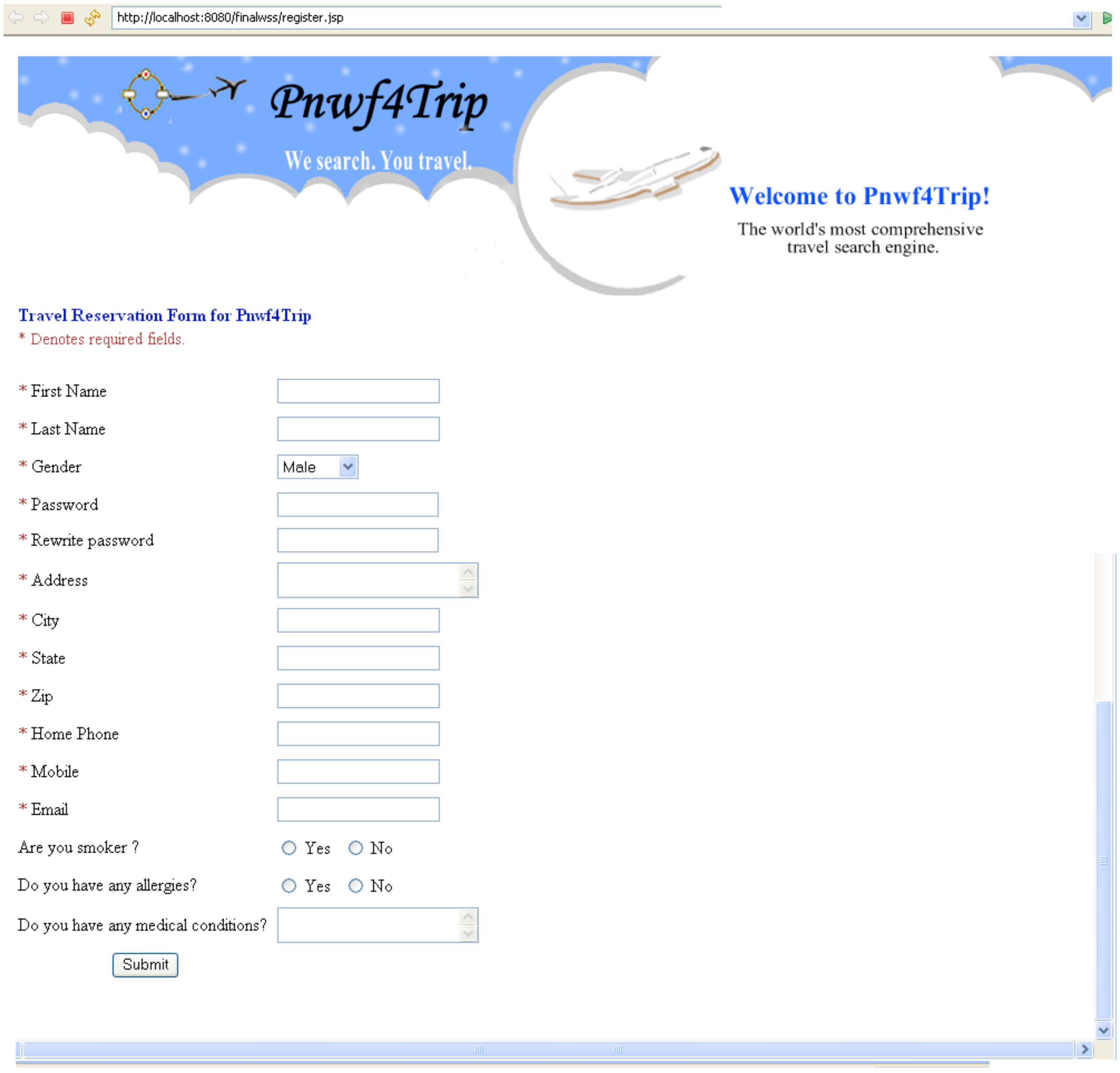
COMPARATIVE STUDY

Discovering and composing individual web services into more complex yet new and receive much attention. Therefore, as shown from the survey in the related research section, a huge number of approaches have been proposed to tackle the problem of web service composition. Most of them are based on either workflow or AI planning techniques. However, the proposed model combines both techniques, workflow and AI planning. Despite all these efforts, establishing web service composition has largely been an ad-hoc, time consuming process, and beyond the human capability to deal with the whole process manually because the web service environment is highly complex and it is not feasible to generate every thing in an automatic way [5]. Table **1** summarizes a comparison between the proposed model and some of the current web service composition approaches. Some features, which have great importance for developing composite services, are identified for the analysis of the composition frameworks [4, 23]. *Service connectivity*: all composition approaches specify how to connect to the service and reason about its inputs and outputs. *Execution monitor*: to monitor and trace service execution. *QOS modeling*: most approaches neglect specification of non-functional QoS properties such as security, dependability, performance, or user's preferences. *Service definition*: specifies which language is used to define a service.

It should be noted that the time that is consumed by the different system components proportional to the execution of web services. Therefore the performance of the proposed model is depended totally on the selected web service execution. Further more, since the main system actors are its web services, the system can apply many number of requests based on these actors.

CASE STUDY

FPN4WSC is a formal model for description and evaluation of web service composition process. FPN4WSC is applied on the travel reservation system as a case study.


http://localhost:8080/finalwss/register.jsp
Pnwf4Trip
We search. You travel.
Welcome to Pnwf4Trip!
The world's most comprehensive travel search engine.
Submit

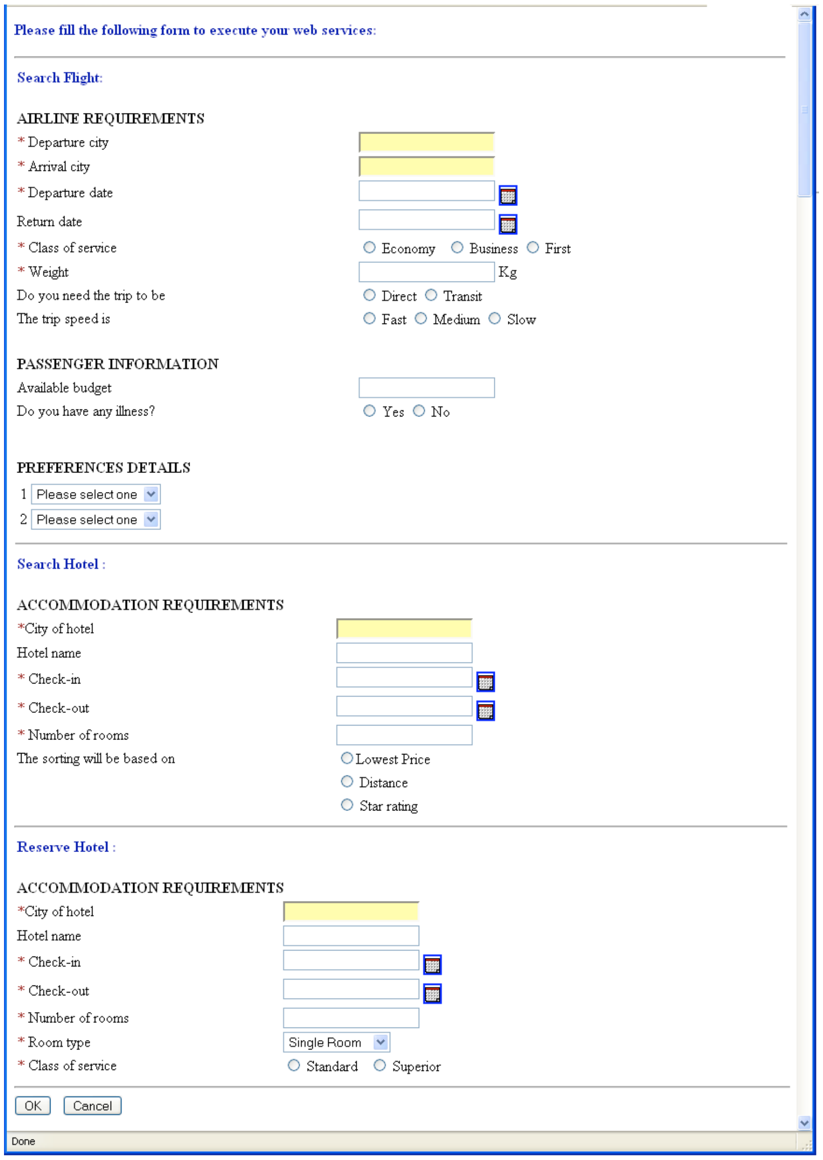
Travel Reservation Form for Pnwf4Trip
* Denotes required fields.

* First Name	
* Last Name	
* Gender	Male
* Password	
* Rewrite password	
* Address	
* City	
* State	
* Zip	
* Home Phone	
* Mobile	
* Email	
Are you smoker ?	Yes No
Do you have any allergies?	Yes No
Do you have any medical conditions?	

Fig. (4). Registration form.

Table 1. Comparison Between FPN4WSC and the Available Web Service Composition (WSC) Frameworks

Framework	Service Connectivity	Composition Strategy	Execution Monitor	QoS Modeling	Service Definition	Graph Support
BPEL4WS	√	workflow composition	×	×	WSDL	×
E-flow	√	workflow composition	√	×	WSDL	√
Petri net based algebra for WSC	√	workflow composition	×	√	Not defined	√
SRN	√	workflow composition	×	√	WSDL	×
Automatic WSC using SHOP2	√	AI composition	√	×	DAML-S	×
WSC based on Fuzzy Petri net and Semantic Web	√	workflow composition	×	√ Fuzzy Petri net	fuzzy Horn clause	×
FPN4WSC	√	workflow and AI composition	√	√ Fuzzy engine	WSDL/OWL-S	√



Please fill the following form to execute your web services:

Search Flight:

AIRLINE REQUIREMENTS

* Departure city	
* Arrival city	
* Departure date	
Return date	
* Class of service	○ Economy ○ Business ○ First
* Weight	Kg
Do you need the trip to be	○ Direct ○ Transit
The trip speed is	○ Fast ○ Medium ○ Slow

PASSENGER INFORMATION

Available budget	
Do you have any illness?	○ Yes ○ No

PREFERENCES DETAILS

1 Please select one

2 Please select one

Search Hotel :

ACCOMMODATION REQUIREMENTS

*City of hotel	
Hotel name	
* Check-in	
* Check-out	
* Number of rooms	
The sorting will be based on	○ Lowest Price ○ Distance ○ Star rating

Reserve Hotel :

ACCOMMODATION REQUIREMENTS

*City of hotel	
Hotel name	
* Check-in	
* Check-out	
* Number of rooms	
* Room type	Single Room
* Class of service	○ Standard ○ Superior

Fig. (5). An empty input form.

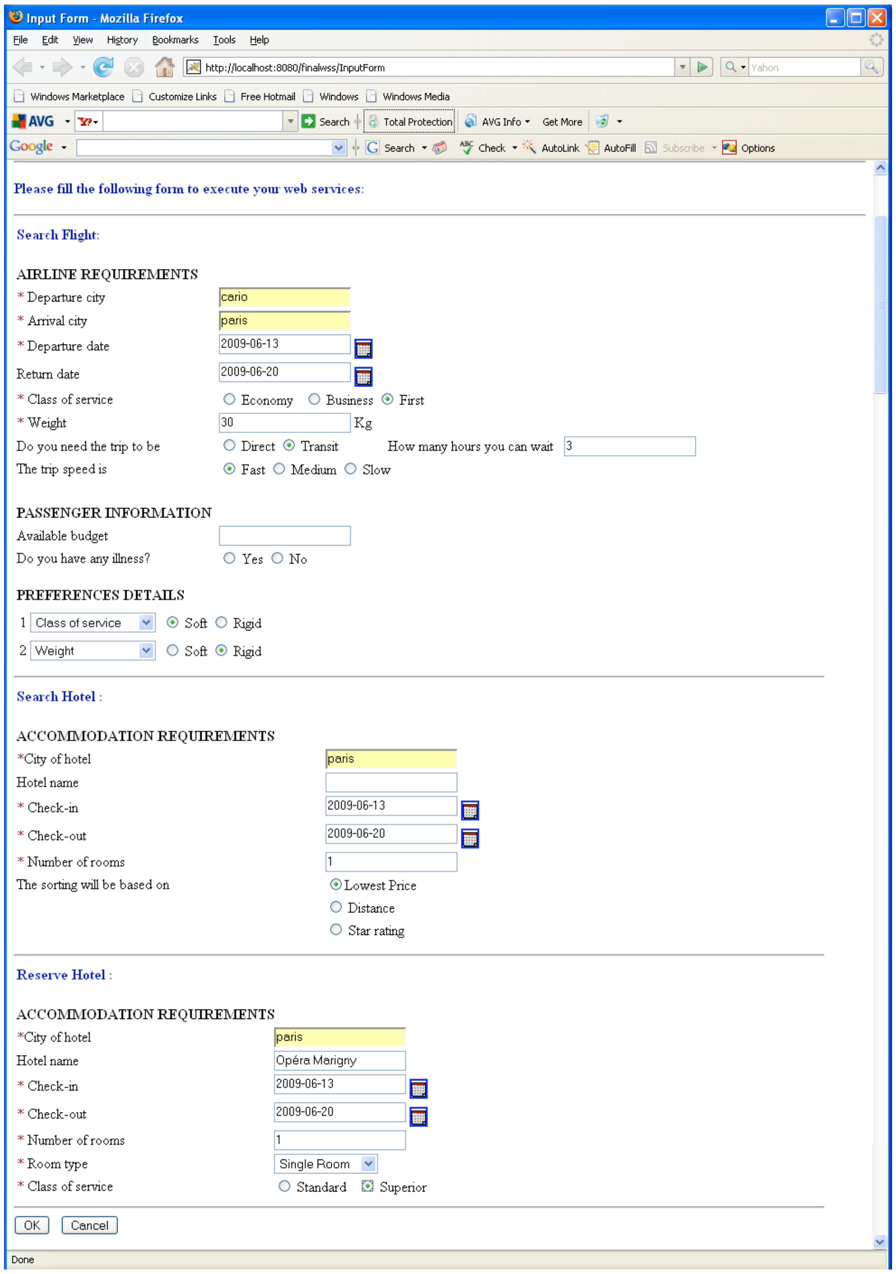
Although many alternatives are available for travel reservation, the FPN4WSC is the most suitable one because it provides with two main added values. First one, it provides the service requester with set of suitable plans for doing the required composition. Second one, it provides the user with uncertainty values for each web service. Further more, it has a very easy GUI.

Below are set of the snapshots which are implemented for testing the model.

Firstly, the user needs to register before using the system functionality. Fig. (**4**) clarifies the registration form. Since the proposed model is applied on the travel reservation, some of the questions depending on this domain exist in the registration form. Also, there is a set of information about the user's health such as if he has any medical conditions, any allergies, or if he is smoking. This information helps the proposed model to get to the user the most suitable trip according to his health.

After the user registers in the system, he can use the system's functionality and run the model to compose his web services. To allow the user to compose set of web services, then he needs to specify a Petri net diagram for the required web services. For example, the diagram in Fig. (**2**) specifies three web services, namely SearchFlight, SearchHotel, and ReserveHotel, to be composed as a new web service such that the first two web services run concurrently and after they end the execution, the final web service starts its execution.

Now the user is ready to physically start executing the composition process by running the FPN4WSC model. The


Input Form - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/finalwss/InputForm
Done

Please fill the following form to execute your web services:

Search Flight:

AIRLINE REQUIREMENTS

* Departure city	cario
* Arrival city	paris
* Departure date	2009-06-13
Return date	2009-06-20
* Class of service	Economy Business [✓ First]
* Weight	30 Kg
Do you need the trip to be	Direct [✓ Transit] How many hours you can wait 3
The trip speed is	[✓ Fast] Medium Slow

PASSENGER INFORMATION

Available budget	
Do you have any illness?	Yes No

PREFERENCES DETAILS

1	Class of service	[✓ Soft] Rigid
2	Weight	Soft [✓ Rigid]

Search Hotel :

ACCOMMODATION REQUIREMENTS

*City of hotel	paris
Hotel name	
* Check-in	2009-06-13
* Check-out	2009-06-20
* Number of rooms	1
The sorting will be based on	[✓ Lowest Price] Distance Star rating

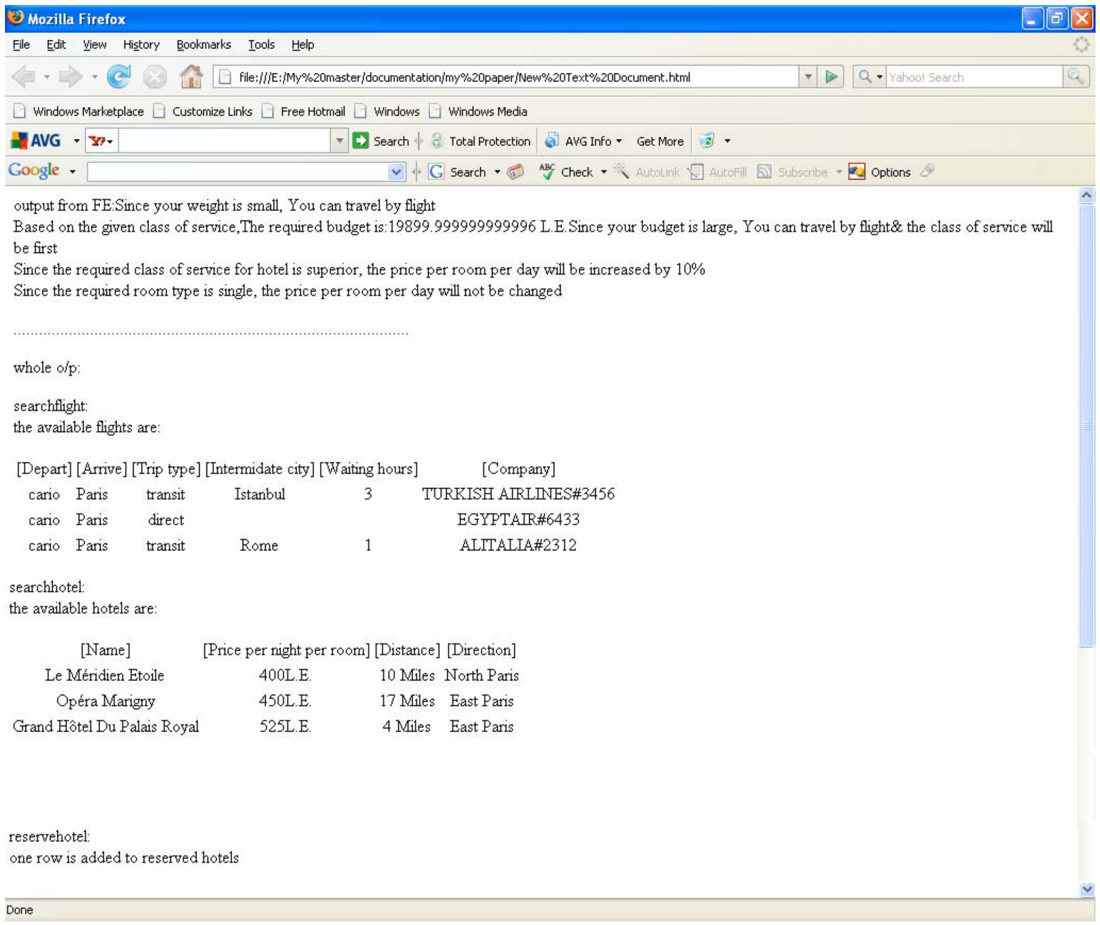
Reserve Hotel :

ACCOMMODATION REQUIREMENTS

*City of hotel	paris
Hotel name	Opéra Marigny
* Check-in	2009-06-13
* Check-out	2009-06-20
* Number of rooms	1
* Room type	Single Room
* Class of service	Standard [✓ Superior]

OK Cancel

Fig. (6). The input form is filled.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///E:/My%20master/documentation/my%20paper/New%20Text%20Document.html
Done

```
output from FE:Since your weight is small, You can travel by flight
Based on the given class of service,The required budget is:19899.999999999996 L.E.Since your budget is large, You can travel by flight& the class of service will
be first
Since the required class of service for hotel is superior, the price per room per day will be increased by 10%
Since the required room type is single, the price per room per day will not be changed

.........................................................................................

whole o/p:

searchflight:
the available flights are:

 [Depart] [Arrive] [Trip type] [Intermidate city] [Waiting hours]        [Company]
    cario    Paris      transit        Istanbul              3        TURKISH AIRLINES#3456
    cario    Paris      direct                                              EGYPTAIR#6433
    cario    Paris      transit          Rome                1                ALITALIA#2312

searchhotel:
the available hotels are:

            [Name]              [Price per night per room] [Distance] [Direction]
      Le Méridien Etoile                  400L.E.           10 Miles  North Paris
        Opéra Marigny                     450L.E.           17 Miles   East Paris
 Grand Hôtel Du Palais Royal              525L.E.            4 Miles   East Paris



reservehotel:
one row is added to reserved hotels
```

Fig. (7). The results of the composed web services.

model tries to parse the given Petri net diagram and generate a vector of concurrent and sequential web services. Then it uses this vector to formulate a lisp problem to be planned later by JSHOP2. Depending on the required web services, the system generates a new web form for the required data of each web service. This web form is displayed to be filled by the user. The snapshots in Figs. (**5**) and (**6**) depict an input form (before and after filling the required data) based on the web services which are specified in the Fig. (**2**). For each web service, the model asks the user about set of data, for example, in the SearchFlight web service, the model asks him about class of service, trip type, trip speed, and the budget. Also it asks him to select the first and the second preferences and which of them is rigid or soft. All these info is used later by the fuzzy engine to get the most suitable trip for the user. In the ReserveHotel web service, the model asks him about the class of service and the room type and by this information the fuzzy engine tries to specify the price of the requested hotel.

After the user fills the input form, he can click on the Ok button to complete the execution of the composition process. In this part, the model tries to use the vector of web services to create a lisp problem and then send it to JSHOP2 to generate a plan for this problem. After JSHOP2 generates a plan for the given problem, the model runs the discovery engine to discover if the required web services are available. If the discovery component succeeds, the fuzzy engine runs to get the best solution for the required web services. Finally, the process execution engine executes the generated plan by running the OWL-S file of each web service and the results of the composed web services are displayed in Fig. (**7**).

CONCLUSION AND FUTURE WORK

Web service composition provides new services for web-based cooperation and it becomes more interested. To describe and model the process of web service composition, a new Petri-net-based model for web service composition is proposed with two new components (SHOP2 and fuzzy engine), i.e. FPN4WSC. The proposed model uses static fuzzy rules which depend on the domain and in the future work the fuzzy rules will be managed in a dynamic way. This problem can be dealt with using the genetic algorithm to generate the fuzzy rules dynamically.

REFERENCES

[1] Y. Tang, L. Chen, K.-T. He, and N. Jing, "SRN: An Extended Petri-Net-Based Workflow Model for Web Service Composition," in *IEEE International Conference on Web Services*, 2004, p. 591.

[2] R. Hamadi, and B. Benatallah, "A Petri-Net-Based Model for Web Service Composition," in *Proceedings of the Fourteenth Australasian Database Conference*, Adelaide, Australia, 2003, pp. 191-200.

[3] M. Champion, C. Ferris, E. Newcomer, and D. Orchard, "Web Services Architecture," *w3.org*, Nov. 14, 2002. [Online]. Available: http://www.w3.org/TR/2002/WD-ws-arch-20021114/ [Accessed: Sept. 7, 2009].

[4] S. Dustdar and W. Schreiner, "A survey on web services composition," *International Journal of Web and Grid Services*, vol .1, no. 1,

pp. 1-30, August 2005. [Online]. Available: http://www.infosys.tuwien.ac.at/Staff/sd/papers/A%20survey%20on%20web%20services%20composition_Dustdar_Schreiner_inPress.pdf [Accessed: Sept. 7, 2009].

[5] J. Rao, and X. Su, "A survey of automated web service composition methods," in *Proceeding of the First International Workshop on Semantic Web Services and Web Process Composition*, San Diego, USA, 2004, pp. 43-54.

[6] "UDDI," [Online]. Available: http://uddi.xml.org/ [Accessed: Sept. 7, 2009].

[7] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1," *w3.org*, Mar. 15, 2001. [Online]. Available: http://www.w3.org/TR/2001/NOTE-wsdl-20010315 [Accessed: Sept. 7, 2009].

[8] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H.F. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP) 1.1," *w3.org*, May 8, 2000. [Online]. Available: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ [Accessed: Sept. 7, 2009].

[9] "Petri net," [Online]. Available: http://en.wikipedia.org/wiki/Petri_net [Accessed: Sept. 7, 2009].

[10] L. Bing and C. Huaping, "Web Service Composition and Analysis: A Petri-net Based Approach," in *First International Conference on Semantics, Knowledge and Grid*, 2005, p. 111.

[11] D. W. Evren, 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, 2003. [Online]. Available: http://www.mindswap.org/papers/ICAPS03-SHOP2.pdf [Accessed: Sept. 7, 2009].

[12] O. Ilghami, "Documentation for JSHOP2," Technical Report CS-TR-4694, Department of Computer Science, University of Maryland, Feb. 2005.

[13] "Common Lisp Language Overview," *lispworks.com*. [Online]. Available: http://www.lispworks.com/products/lisp-overview.html [Accessed: Sept. 7, 2009].

[14] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, "Business process execution language for Web services 1.1," Technical report, May 2003. [Online]. Available: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf [Accessed: Sept. 7, 2009].

[15] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. Mcllraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, "OWL-S: Semantic Markup for Web Services," *w3.org*, Nov 22, 2004. [Online]. Available: http://www.w3.org/Submission/OWL-S/ [Accessed: Sept. 7, 2009].

[16] F. Casati, S. Ilnicki, L.-j. Jin, V. Krishnamoorthy, and M.-C. Shan, "Adaptive and Dynamic Service Composition in eFlow," in *Proceedings of the 12th International Conference on Advanced Information Systems Engineering*, 2000, pp. 13-31.

[17] R. Fu, W. Dong, G. Yang, Y. Mei, and X. Dong, "Fuzzy Petri Net-Based Optimized Semantic Web Service Composition," in *Proceedings of the 2008 Seventh International Conference on Grid and Cooperative Computing*, 2008, pp. 496-502.

[18] "Home page- SHOP2," [Online]. Available: http://www.cs.umd.edu/projects/shop/index.html [Accessed: Sept. 7, 2009].

[19] "OWL-S API," *mindswap.org*, [Online]. Available: http://www.mindswap.org/2004/owl-s/api/download.shtml. [Accessed: Sept. 7, 2009].

[20] T. Freytag, and A. Eckleder, "Workflow Petri net Designer (WoPeD)," *woped.org*, [Online]. Available: http://www.woped.org/ [Accessed: Sept. 7, 2009].

[21] "A Java Fuzzy Engine Example," [Online]. Available: http://people.clarkson.edu/~esazonov/neural_fuzzy/loadsway/LoadSway.htm [Accessed: Sept. 7, 2009].

[22] "Browse SHOP Files on SourceForge.net," *sourceforge.net*. [Online]. Available: http://sourceforge.net/projects/shop/files/ [Accessed: Sept. 7, 2009].

[23] N. Milanovic, and M. Malek, "Current solutions for web service composition," *IEEE Internet Computing*, vol. 8, pp. 51-59, 2004.

Received: September 10, 2009 Revised: November 25, 2009 Accepted: December 16, 2009