SOFTWARE ARCHITECTURAL PATTERNS FOR SERVICE COMPOSITION

Ghadeer Ghazal[1], Amjad. Hudaib[2], Waffa Maitah[3]

[1,2]Department of Computer Information Systems, Faculty of King Abdullah II, The University of Jordan, Jordan
[3] Department of Computer Science, Faculty of Information Technology, Al al-Bayt University, Jordan

ABSTRACT

Service-oriented computing is meant to support loose relationships between organizations; Service-oriented architectures often have the goal to integrate various distributed services of one or more organizations in a flexible way to be able to quickly react on business changes.

Distributed services provided a new way of distributed computing that achieve the interoperability between heterogonous application through platform and language independent interfaces. The creation of value added services by composition of existing ones is gaining a significant momentum. Distributed service composition is meant to support loose relationships between implemented services in order to provide new functions. A composite service is the one resulting from the integration, coordination and synchronization of different service components. In this paper, we generated A Services Composition Model (SCM) that provides a general solution for the services composition problem by realizing the requirements of a new service using the requirements of the already existing service. We explained in details all the steps of the composition process; services registration, services discovery, services selection, services invoking, and services integration. Although the SCM is not bounded to one particular algorithm to compose services, we generated an application as an example to test our Service Composition Model.

We also generated the Services Composition Language (SCL) as a simple text-based language which allows the user to express the requirements of his request, the inserted request will then be analyzed using our Parsing Algorithm to determine the name of the requested services, after that our Service Composition Algorithm will execute all the steps of the composition process and return the result of the composition to the user.

KEYWORDS

Services, Services Composition, Architectural Pattern, Services Composition Model (SCM).

1. INTRODUCTION

Services offered online are defined as a loosely coupled, reusable software component that encapsulates discrete functionality. One essential characteristic of such services is the ability to be described, published, discovered and invoked dynamically in a distributed computing environment. These services can range from simple calculations and data retrievals to complex business applications (Sommerville, 2007), (Agarwl, et al., 2008).

In this research, we attempt to enhance the performance of Service Composition, which aims at increasing reusability, flexibility and maintainability by breaking down the functionalities into services running platform-independently and exhibiting their interface with a description of functionality and the domain of objects employed.

Another significant point of services composition is that composite services must not be arranged at design time. The final functionalities of services working together are defined by the service request and must not be specified beforehand. When composing services, a composition only needs to include a reference that identifies what service will be invoked, the dynamic binding takes place as late as possible when the service is actually invoked.

There are situations in which a client request cannot be satisfied by any single available service, but a composite service obtained by combining some available services might fulfill such a request. The services used to form a composite service are referred to as component services.

1.1 SERVICES

Services provide a new way of distributed computing that achieves the interoperability between heterogonous application through platform and language independent interfaces. The creation of a value which has added services by composition of the already existing ones is gaining a significant momentum. (Guoping, et al., 2008) t (Rao, et al., 2005).

1.2 ARCHITECTURAL PATTERNS

Software architectural pattern is a description of the architectural design problem and the essence of its solution, (Hudaib and Montangero, 2002), (Cooper, 1998) and (Tut and Edmond, 2002). Software architectural pattern has the following essential elements (Gamma, et al., 1998).

- Pattern Name: A meaningful name to refer to the pattern, typically a single word or short phrase.
- A description of the problem: with a concrete example and its context, (Al-Masri,et al., 2008). It also describes a solution to the concrete example and when to use the pattern.
- General solution: describes the elements of a design, their relationships, responsibilities and collaborations.
- Consequences: good and bad of using the given solution. It describes the results and trade-off of applying the pattern.
- Related patterns: naming patterns that are closely related (Buschmann, et al, 2007).

1.3 Automated Web Service Composition

Automated distributed Service Composition denotes a method to integrate services at runtime. The complete service runs by composing components based on a service request. The service request specifies the requested service, which does not exist before issuing the request. (Gekas, et al., 2005).

2. LITERATUE REVIEW

2.1 SOA Structure

Keen, et al. (2004) interested in SOA; the basic assumption of SOA is that there are many consumers that require services. Consumers are also referred to as clients or customers; there are many providers that provide services on the network. These two groups have to be linked together in a dynamic way.

Service providers register their services at the broker (Chmielowiec, et al., 2008); service consumers request a service from the service broker, which returns a known provider for the requested service. Consumer and provider agree on the semantics. The consumer then binds himself to the service provider and uses the service.

2.2 Services Composition Models

Stein, et al. (2009) defined the workflow of the service model and the services that it can provision in order to execute these workflows and describe the lifecycle Figure (2.2) shows the Workflow Lifecycle for the service model.

(1) Workflow Selection. a general workflow is chosen to suit the consumer's current objectives. This is generally created either manually by domain experts or automatically by a planner.
(2) Matching. Once a general workflow has been selected, abstract tasks are mapped to candidate service instances via a matching process. Here, the consumer searches a public service registry or requests matching services from a broker.
(3) Provisioning. Given lists of matching services, the consumer provisions individual service instances for each task of the workflow.
(4) Invocation. When appropriate services have been provisioned, the consumer starts to invoke the chosen services as dictated by the ordering constraints of the workflow. If services fail to complete their tasks, the consumer may provision other services, until the workflow is successfully completed.

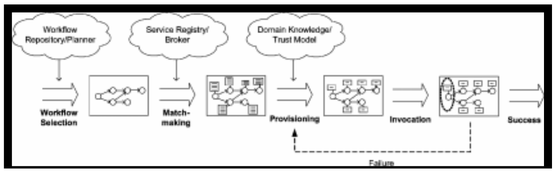


Figure (2.2): Lifecycle of Service Model (Stein, et al., 2009)

2.3 Patterns used in Service Composition

Barrett, et al. (2006) provided a subtle difference between two of the modelling aspect within the service composition, namely workflow and distribution patterns. They considered workflows as compositional services, whereby the internal and external messages to and from services are modelled. In contrast, distribution patterns are considered composition where only the external messages between services are modelled. This approach considers the basic four distributed patterns: Centralized, Decentralized or peer-to-peer, Ring, Hierarchical.

Zirpins, et al. (2004) addressed using the patterns through the interaction composition. This approach aimed to gather generic interaction patterns in a catalogue, this catalogue contains the concrete generic interaction patterns, coordination policies and coordination idioms and their relationships with the services to refines the service based on analysis of its interaction patterns.

3. Model and design

3.1 Services Composition Model Overview

Services Composition Model (SCM) aims to provide a general solution for the services composition problem by realizing the requirements of a new service using the requirements of the existing service and to automate the steps of the composition process; services discovery, services selection, services invoking and services integration.

Although the SCM is not bounded to one particular algorithm, we generate an example to test the SCM.

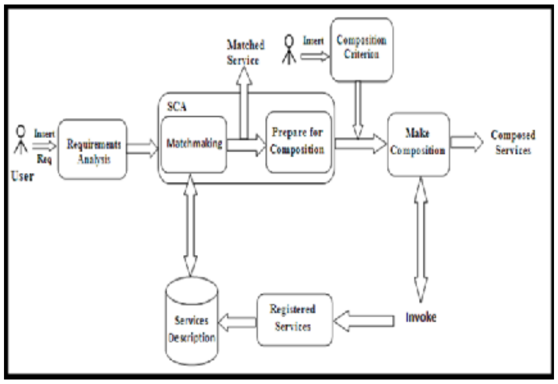


Figure (3.1) shows a brief description of the stages of the services composition model

Users: A user may compose some distributed services that may belong to different administrative domains. Each user inserts the request of service using the service composition language as an input language, the requirements of the requested service may be satisfied by one of the following cases:

1. **Matched service**: there is one service that satisfies the user request, or
2. **Composed services**: the request could be satisfied by composing more than one service. In this case we need the service composition algorithm to discover, select and compose these services.

•**Requirements Analysis**: We introduce a new algorithm (Parsing Algorithm) to analyze the inserted request from the user to specify the name of the services that satisfy the user's request.
•**Services Descriptions**: The description of each service generated and stored in the database; this database will be used to discover and select the desired service.
•**Service Composition Algorithm (SCA)**: Services Composition Algorithm discovers and selects the most suitable services to be integrated; this selection will be according to the inserted requirements and composition criterion from the user, the SCA work in three steps:

1. **Matchmaking**: Once the requirements analyzed; the services composition algorithm discovers the services by matchmaking between inserted requirements and the description of the available services, this will generate abstract services.

2. **Prepare for composition**: Contains the entire abstract services that are chosen to suit the consumer's requirements; there are many services which may satisfy one requirement. This is generally created automatically allowing the consumer to retrieve services from a repository then according to the inserted composition's criterion from the user, the composition algorithm will select the actual services from the abstract services.

3. **Make composition**: According to the inserted composition's criterion from the user, the composition algorithm will select the actual services from the abstract services to be invoked and integrated based on WSDL. This step will invoke and integrate the selected services from the previous ste..

• **Registered Services**: Each provider can register his service by implementing the DLL and each service contains the Web Services Description Language file (WSDL) as an indicator for the services methods and functionalities.

In The following sections, we will discuss the stages of the Services Composition Model in details.

3.2 Services Registry

Services registry is a searchable directory that contains a specification about the available services from the provider. We suggest classifying these service's specifications to be stored in different categories in the centralized repository; each category contains all the services that provide the same function.

3.2.1 Classified Repository pattern

In the following section we propose a new pattern to classify the registered services.

The structure of the classified repository pattern includes four participants as depicted in Figure (3.2):

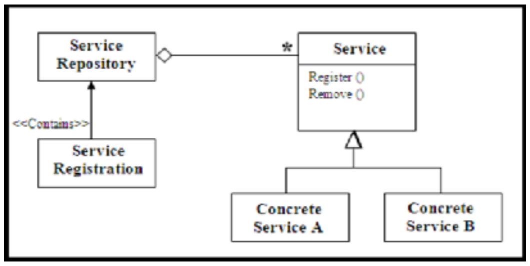


Figure (3.2): components of Classified Repository Pattern

3.2.2 DYNAMIC LINK LANGUAGE

We also suggest the idea of using the Dynamic Link Language (DLL) as an abstract interface. It defines a unique service name, function name parameters and return value as follows:

```
Public interface service name
Function name (parameters of the function)
Return value
End interface
```

One DLL will be implemented by all the services which provide the same function and stored in the same category, for example the {S11, S12, S13 … S1n} are the services that stored in (category 1) and provides the same function, these services were implemented from the same DLL (DLL1) and stored in one layer as depicted in Figure (3.3).

$$\left[\begin{array}{c|cccccc} Dll_1 & S_{11} & S_{12} & S_{13} & \dots\dots & S_{1m} \\ Dll_2 & S_{21} & S_{22} & S_{23} & \dots\dots & S_{2m} \\ Dll_3 & S_{31} & S_{32} & S_{33} & \dots\dots & S_{3m} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ Dll_n & Sn_1 & S_{n2} & S_{n3} & \dots\dots & S_{nm} \end{array}\right]$$

Figure (3.3): DLL and related services.

3.3 Service Composition Language (SCL)

We propose a new Service Composition language (SCL) for the composition purpose, which allows the user to express his service request in a simple way using the SCL notations.

Service Composition language is a simple text-based language. It absorbs most of the syntax from VB.net and deals with the services that should be requested to be composed. The SCL is considered a simpler composition language than the available composition BPEL language.

The main aim of the SCL is to automate the processes of the services selection and invoking. BPEL is a powerful XML used in services composition, but the selection and the invoking of the services in this language is static, so we generate the SCL which is able to select and invoke the services dynamically.

3.4 Requirements Analysis

The inserted requirements (service composition program) will be analyzed by the parsing algorithm. This algorithm is able to analyze the inserted program to generate the name of the requested services and determine how the services will be composed. The pseudo code of the parsing algorithm is depicted in Figure (3.4)

```
Input: composition program
Instruction = first line
While instruction ≠ final line
Get next line
    If instruction contains (if statement)
    Parse if statement
    Else if it contains (while statement)
    Parse while statement
    Else if it contains (variable, and assignment operator)
    Parse variable statement
    Else if it contains (service calling)
    Invoke the services composition algorithm
    End if
End while
```

Figure (3.4): Parsing Algorithm Pseudo code

The parsing algorithm contains the following two major functions:

1.Parse Function parses each line of the inserted composition language and determines if the instruction contains a variable, a return message, a service calling, or an if statement.
2. Resolve Variable trims all the external notations such that ($, $$, $$$) and invokes the Services Composition algorithm.

3.5 Services Description

The services description is a description for registered service. Such a description would be inserted in the database to allow the user to discover and select the desired services.
The description for each service is stated in Figure (3.5).

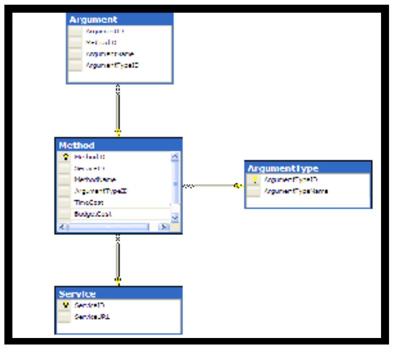


Figure (3.5): Attributes to describe the services.
This description of the registered web services will be as the follow:

• Information about services: Services ID and service URL.
• Information about functions (methods) in the service: method ID, method name, Criteria of the method (budget cost, time cost).
• Information about the arguments in the method: argument ID, argument type.

3.6 Services Composition Algorithm

Services Composition Algorithm is the actual algorithm performing composition upon the user's request. The SCA composes the services thorough many steps as shown in Figure (3.6); (1) matchmaking stage: the requirements are matched to the service's description, (2) preparation stage: the most suitable services from the matched services are selected relying on the composition criterion and then prepared for the next stage known as composition, (3) composition stage: the matched services are finally integrated and composed.

The Services Composition Algorithm depends on runtime discovery and selection for the registered services

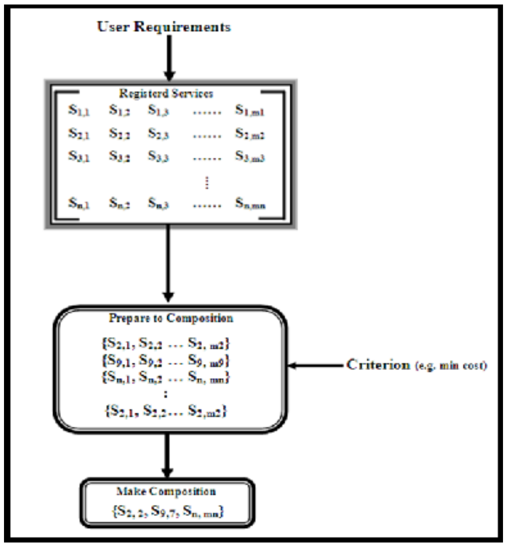

User Requirements
Registerd Services
$S_{1,1}$ $S_{1,2}$ $S_{1,3}$ ...... $S_{1,m1}$
$S_{2,1}$ $S_{2,2}$ $S_{2,3}$ ...... $S_{2,m2}$
$S_{3,1}$ $S_{3,2}$ $S_{3,3}$ ...... $S_{3,m3}$
$S_{n,1}$ $S_{n,2}$ $S_{n,3}$ ...... $S_{n,mn}$
Prepare to Composition
$\{S_{2,1}, S_{2,2} \dots S_{2,m2}\}$
$\{S_{9,1}, S_{9,2} \dots S_{9,m9}\}$
$\{S_{n,1}, S_{n,2} \dots S_{n,mn}\}$
$\{S_{2,1}, S_{2,2} \dots S_{2,m2}\}$
Criterion (e.g. min cost)
Make Composition
$\{S_{2,2}, S_{9,7}, S_{n,mn}\}$

Figure (3.6): Composition process

3.6.1 Matchmaking

Matchmaking is the process of services discovery of the registered services (Al-Masri, et al., 2008); the description of the desired service and the description of the available services which have to be compared. If the available services match, the required service is proved to be successfully discovered.

The matchmaking step generates all the services that match the inserted requirements as abstract services to prepare for composition; for example ,the abstract services for two requested services are the following services {(S11, S12…S1n) and (S91, S92…S9n)}, these services generated from the available services.

3.6.2 Prepare for Composition

This phase enables the selection of the services from the abstract services according to the preferred composition criterion; the non functional requirements which inserted by the user (e.g. cost) compared with the values of the abstract services to generate the concrete services to make the composition, these services are really invoked and integrated e.g. {S11, S49, Snm}.

3.6.3 Services Composition Criteria

Multiple distributed services may provide similar functionalities with different non-functional property values (e.g., different prices). To select the services that will be invoked, we suppose many criteria for the composition.

In our example, we consider three generic quality criteria for composition process (1) execution price, (2) execution duration and (3) reliability.

•Execution price (cost): the execution price is the amount of money that a service requester has to pay for executing the operation.
•Execution duration: the execution duration measures the expected delay in seconds between the moment when a request is sent and the moment when the results are received.
•Reliability is the probability that a request is correctly responded within a maximum expected time frame (which is published in the service description) (Zeng, et al. 2003).

3.6.4 Make Composition

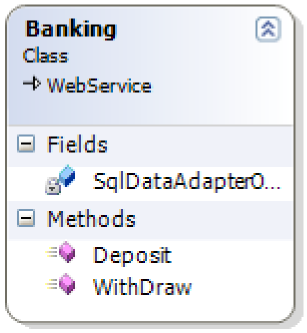
This step invokes and integrates the selected services from the previous step. The composition algorithm will invoke the selected services based on the WSDL.

4. IMPLEMENTATION

We develop (6) services in our Application; Banking service, Police service, Average Calculation Service, Grade Point Average (GPA) Service, File Search Service and Get File Attributes Service. The first step is to develop these Web Services and deploy them. The actual code is written in VB.Net classes.

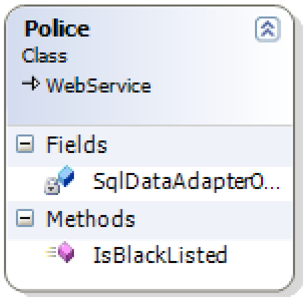
The description and deployment of the Web Services are accomplished with WSDL documents.

1. Banking Service allows the client of the bank who has an account to interact with the service to carry out certain transactions online, such a withdraw transaction is given in Figure (4.1).



Figure(4.1): Banking Service

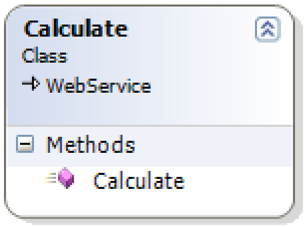
2. Police service: this service examines if the credit card of the user is blocked or not. The Police Service has one main function called IsBlackListed
    IsBlackListed as shown in Figure (4.2).



Figure(4.2): Police Service

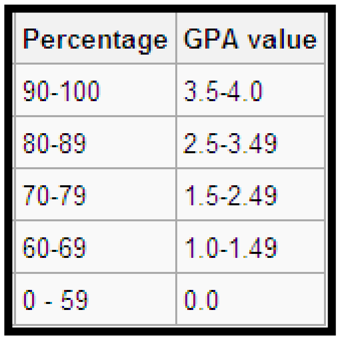
3. Average Calculate Service: calculate the average of the student marks using the following formula:

Mean = sum of marks / number of marks
The Average Calculate Service has one function named with Calculate as shown in Figure (4.3)



Figure(4.3): Average calculate service

4.GPA Service calculates the Grade Point Average according to the shown standard provided in Figure (4.4).



Percentage	GPA value
90-100	3.5-4.0
80-89	2.5-3.49
70-79	1.5-2.49
60-69	1.0-1.49
0 - 59	0.0

Figure (4.4): GPA Standard.

The GPA service has one function named with GetGPA as show n in Figure (4.5).

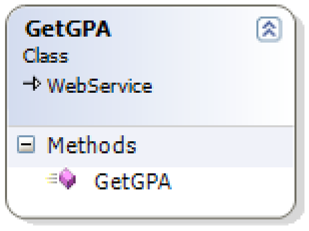


Figure (4.5): GPA Service

5. File Search service: search for file at specific location on the server based on name and extension of the file (e.g. c:\com.doc), the File Search service has one function named with Search File as shown in Figure (4.6).

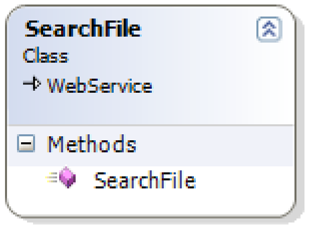


Figure (4.6): File Search Service

6. Get file attributes service retrieves the information about a specific file; this service retrieves the file name, file extension, creation time and size of the file. The Get file attributes service has one function known as GetFileAttributes as shown in Figure (4.7).

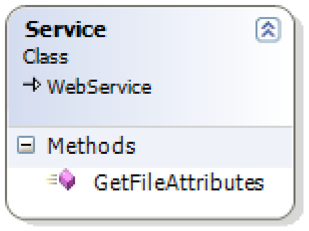


Figure (4.7): Get File Attributes service

4.1 Case Study1 (Payment Scenario)

The following payment example illustrates a possible application for Web Service Composition. A customer consults a web page of online payment service.

The payment service has many business processes; such as (1) check the information of the Credit Card then (2) check if the credit card is black (3) withdraw the inserted amount of the money if the information is correct (4) or deny the process if there is any error in the inserted information.

Payment service is basically composed of two services; the Banking Service and IsBlackListedCreditCard Service as shown in Figure (4.8).

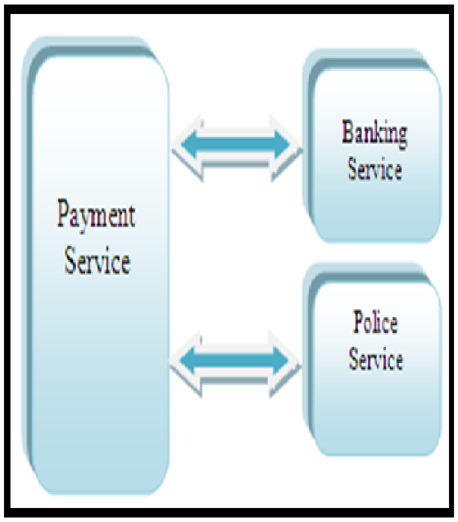


Figure (4.8): Payment Service.

1. The user writes a services request using the composition language, The user uses the (if) statement to compose withdraws function with the is blacklisted service.
2. Parse and analyze the inserted program line by line to specify the requested services using the parsing algorithm.
3. Discover all the services that matched the inserted one by matchmaking between the requested services in composition program and the registered services, the discovery will generate an abstract work flow contains all the payments and isblacklisted services (e.g. payment1, payment2, islacklisted1).
4. Select the most suitable services from the generated workflow according the selected criterion (e.g. payment1 and isblacklisted1). This step will generate the concrete workflow using the GetSuitableClass function.
5. Invoke the selected services based on the WSDL using the WebClient Object; these services will be composed.
6. Integrate the invoked services and return the result for the user.

4.2 Case Study (2)

Another scenario works with two similar Web Services representing composition process use the while composition style.

The file discovery service Figure (4.9) has many services; (1) file search, this service will not stop until it finds the requested file, (2) return the information about the found file.

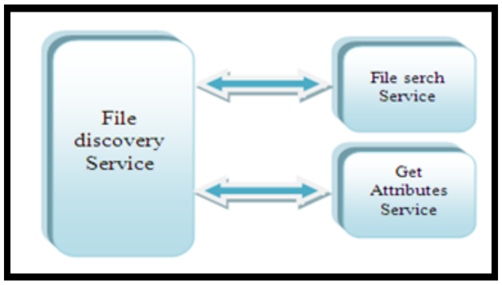


Figure (4.9): File Discovery Service.

The user writes a services request using the composition language. The user used the (while) statement to compose search file service with get file attributes service.

The file search service takes the name, location and extension of a file as input, then search for the file at the specified location. This service will search for the file (while loop) the file is found. After that it invokes the get file attributes to return the information about the file.

5. Comparison between SCM via another Composition Models.

This section aims at providing a comparison between many of the presented models in the related work chapter. This comparison is achieved by creating a table, listing many of the Models and listing some of the most important features that were identified to categorize the composition Models.

The comparison between the presented models will be according to the following categories:

- Composition strategy: Static and dynamic composition strategies concern the time when web services are composed; the Static composition takes place during design-time, while the dynamic composition takes place during run-time. Another composition strategy is the declarative way. The declarative approach consists of two phases: the first phase takes an initial situation and the desired goal as starting point and constructs generic plans to reach the goal; the latter one chooses one generic plan, discovers appropriate services and builds a workflow out of them.
- Execution monitor: The central scheduler controls the execution of the components of the composite service.
- Dynamic Conversation selection: allows the user to select the conversation within a service node at run-time, which may be useful when services are also discovered at run-time.
- QoS Modeling: service discovery in UDDI is not limited to the functional requirements only, but it also considers the nonfunctional requirements QoS to discover and select the services.
- WSDL Language: uses the Web Services Description Language to describe the services functionalities.
- Graph support: Composite services are modeled by a graph defining the flow of service invocations similar to UML activity diagrams.
- Service Composition Language is used by the user to express his requirements for services composition.

Table (5.1): Comparison between Web Services Composition Models

Categories / Models	Composition Strategy	Execution Monitor	Dynamic conversation selection	QoS Modeling	WSDL Language	Graph Support	Service Composition Language
SCM	**Dynamic Composition**	**Yes**	**Yes**	**Yes**	**Yes**	**No**	**Yes**
PAMCS	Dynamic Composition	Yes	Yes	No	No	No	No
SELF-SERV	Declarative Composition	No	No	No	No	Yes	No
E-Flow	Dynamic Composition	Yes	Yes	No	No	Yes	No
FAWSS	Dynamic Composition	No	No	Yes	Yes	No	No

6.1 Summary and Conclusion

Service composition enables flexible process of new application from reuse available services by composing and integrating services.

This paper presents many results:

1. Develop a new model for the service composition problem.
2. Generate a composition language to express the user request and an algorithm (Parsing Algorithm) which is able to analyze the composition program.
3. Generate a new composition algorithm; this algorithm able to discover, select and invoke the services based on WSDL.
4. Generate a new architectural pattern (classified repository pattern) as suggestion to store the registered services.
5. Applying and executing services composition process by providing many case studies as examples; these cases based on the provided services composition approach.
6. Providing solutions for many of the difficulties in the services composition process.
7. Provide a formal description for the services registration process and for the composition process using the Z-Language.
8. Explain the Services Composition Architecture with the Composition Process.
9. Comparison between the Business Process Execution Language (BPEL) and the provided approach.

This research also discussed in details the stages of the services composition model:

1. Services registration: the providers implements the DLL for services registration process, the registered services will be stored in the classified repository pattern.
2. Services description; Web Services Description Language (WSDL) and the description in the database.

3. Services discovery: matchmaking between the inserted requirement and the description of the services.
4. Services selection, matchmaking between the inserted criterion the most suitable services.
5. Services invoking based on the WSDL and the integration of these services.

REFERENCES

1. Sommerville, I. (2007), Software Engineering, 8th edition, united State of America, Pearson Education Limited.
2. Guoping, Z. Huijuan, Z. Zhibin, W (2009), A QoS-based web services selection method for dynamic web service composition, First International Workshop on Education Technology and Computer Science, vol. 3, Wuhan, Hubei, China, 832-835.
3. Hudaib, A. and Montanero, C. (2002), A UML Profile to Support the Formal Presentation of Software Architecture, Proceeding of the 26th IEEE international Computer Software and Application conference on prolonging software life : Development and Redevelopment (COMPSAC'02), 217-223.
4. Cooper, J.W. (1998), The Design Patterns Java Companion, (1st ed.), James W. Cooper.
5. Gamma, E. Helm, R. Johnson, R. and Vlissides, J. (1998), Design Patterns: Elements of Reusable Object-Oriented Software, (1st ed.), Addison-Wesley, Baarn, Holland.
6. Tut, M.T and Edmond, D. (2002), The use of Patterns in Service Composition, Revised Papers from the International Workshop on Web Services, E-Business and the Semantic Web, Vol. 2512, London, UK, 28 - 40 .
7. Agarwl, V. Chafle, G. Mittal, S. and Srivastava, B. (2008), Understanding approaches for web service composition and execution, Proceedings of the 1st Bangalore annual Compute conference, Article No. 1 Bangalore, India.
8. Al-Masri, E. and Mahmoud, Q.H. (2008), Investigating Web Services on the World Wide Web, Proceeding of the 17th international conference on World Wide Web, Beijing, China, April 21–25, 2008, 795-804.
9. Rao, J. and Su, X. (2005), A Survey of Automated Web Service Composition Methods. Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, San Diego, California, USA, Volume: 3387, 43-54.
10. Buschmann, F. Henney, K. and C.Schidt, D. (2007), pattern – Oriented Software Architecture on Patterns and Pattern Language, Vol.5.
11. Gekas, J. and Fasli, M. (2005), Automatic Web Service Composition Using Web Connectivity Analysis Techniques, W3C Workshop on Frameworks for Semantics in Web Services 2005 Position Paper.
12. Keen, M. Acharya, A. Bishop, S. Hopkins, A. Milinski, S. Nott, C. Robinson, R. Adams, J. Verschueren, P. (2004), Patterns: Implementing an SOA Using an Enterprise Service Bus SOA Steps, (1st edition), IBM Corp.
13. Chmielowiec, A. Pierre, G. Gordijn, J. (2008), Technical challenges in market-driven automated service provisioning, Proceedings of the 3rd workshop on Middleware for service oriented computing, Leuven, Belgium, 25-30.
14. Stein, S. Payne, T.R. Jennings, N.R. (2009), Flexible provisioning of web service workflows, Proceedings of the 17th European Conference on Artificial Intelligence, ACM Transactions on Internet Technology, Volume 9, United Kingdom, 295–299.
15. Barrett, R. Pahl, C. M.Patcas, L. and Murphy, J. (2006), Model driven distribution pattern design for dynamic web service compositions, Proceedings of the 6th international conference on Web engineering, Palo Alto, California, USA, July 11-14, 2006, 129 – 136.
16. Zirpins, C. Lamersdorf, W. and Baier,T. (2004), Flexible Coordination of Service Interaction Patterns, Proceedings of the 2nd international conference on Service oriented computing, New York, USA, 49-56.