Int. J. Communications, Network and System Sciences, 2012, 5, 644-660
http://dx.doi.org/10.4236/ijcns.2012.529075 Published Online September 2012 (http://www.SciRP.org/journal/ijcns)

MDCHeS: Model-Driven Dynamic Composition of Heterogeneous Service*

S. Farokhi, A. Ghaffari, H. Haghighi, F. Shams
Automated Software Engineering Research Group, Electrical and Computer Engineering Faculty,
Shahid Beheshti University GC, Tehran, Iran
Email: so.farokhi@mail.sbu.ac.ir, am.ghaffari@mail.sbu.ac.ir, H_haghighi@sbu.ac.ir, F_Shams@sbu.ac.ir

Received June 27, 2012; revised July 24, 2012; accepted July 30, 2012

ABSTRACT

Web Service Composition provides an opportunity for enterprises to increase the ability to adapt themselves to frequent changes in users' requirements by integrating existing services. Our research has focused on proposing a framework to support dynamic composition and to use both SOAP-based and RESTful Web services simultaneously in composite services. In this paper a framework called "Model-driven Dynamic Composition of Heterogeneous Service" (MDCHeS) is introduced. It is elaborated in three different ways; each represents a particular view of the framework: data view, which consists of a Meta model and composition elements as well their relationships; process view, which introduces composition phases and used models in each phase; and component view, which shows an abstract view of the components and their interactions. In order to increase the dynamicity of MDCHeS framework, Model Driven Architecture and proxy based ideas are used.

Keywords: Service-Oriented Architecture; Web Service Composition; RESTful Web Service; SOAP-Based Web Service; Model Driven Architecture; Proxy Service

1. Introduction

Composition is one of the central and most important tenets of service-oriented computing [1]. Web Services Composition (WSC) has evolved much interest amongst the researchers in the academic world as well as the industry. This is because the composite service presents the features that an individual service cannot present. Indeed WSC provides the opportunity for enterprises to adapt themselves quickly to frequent changes in user demands via integrating of existing services[1].

There are currently two main categories of Web services: services that are developed based on traditional, heavy weight SOAP protocol; and services that are developed based on REST protocol, which is newer and simpler. The Web services in first category were more common in the past. They are described by Web Service Description Language (WSDL) and are referred as SOAP based services in this paper. On the other hand, the REST architectural style, corresponding to the RESTful services, is emerging as an alternative technology platform to realize service-oriented architecture [2]. RESTful Web services introduce a new level of abstraction, which does not fit well with the message-oriented paradigm of WSDL [3]. Although SOAP based services are considered as asset of most of the current organizations, because of the simplicity of RESTful services, they are widely accepted by the public and with the goal of attracting a larger user community, more and more service providers are switching to use them [4]. The existence and need of these two different kinds of Web services has brought up new challenges in WSC area. The main question is how to compose heterogeneous Web services in a single composite service.

In addition, Web service environment is highly dynamic in nature and the number of Web service providers is constantly increasing that it is leading to the availability of new services in daily basis [5]. In such a dynamic environment, a dynamic WSC approach is required.

Besides, Model driven Architecture (MDA) facilitates development of WSC and decreases its complexity by providing different abstraction levels. It leads to separating the composition logic from the specific composition implementation [6] achieved by introducing Platform Independent Model (PIM), Platform Specified Model (PSM), and transformation between them automatically. Therefore, by assisting of MDA, Web service design can be captured by PIM, which focuses on the business logic

*The preliminary version of this paper was presented at International Conference on Information Science and Applications, ICISA 2011, Jeju Island, Republic of Korea.

[1]The terms "service" and "Web service" have been used interchangeably in this paper.

rather than the underlying platform technologies. On the other hand, Web service implementation is presented as PSM, which is related to certain platforms and business process description languages [7].

WSC has triggered a considerable number of academic and industrial research efforts that each of them focuses on certain phases of the composition process. Based on a survey on existing approaches [8], which some of them are discussed in Section 2, some shortcomings of current approaches have been identified as follows:

- Since the number of available services, has been dramatically increasing over the Web during recent years, WSC procedure has become a highly complex challenge, but most methods do not offer a direct solution to address this issue.
- According to the frequent changes in the Web service environment, static WSC methods suffer from adapting to those changes.
- Most of the current methods are not applicable on composing heterogeneous Web services.

To overcome mentioned problems, we proposed a framework called MDCHeS (Model-driven Dynamic Composition of Heterogeneous Service). It covers the whole phases of designing composite services and has the required elements and components as well. We utilized the Model Driven Architecture and Proxy concepts to increase the dynamicity and adaptability of our framework.

The rest of this paper is structured as follows: Related work is presented in Section 2. The proposed framework introduced in Section 3. In order to evaluate our framework, we have three approaches in the Evaluation section, first it will be evaluated by performing a case study, second an expert questionnaires related to MDCHeS framework will be represented and third a comparison with some other leading frameworks has been done. The last section is dedicated to the conclusion and future work in this field.

2. Related Work

In this section, a brief review of some service composition methods and summary of some shortcoming, which are addressed in our work, are presented. For a better understanding, they are categorized according to different characteristics of our proposed method.

2.1. Heterogeneous Service Composition

In [9] as one of the initial attempts to integrate SOAP and RESTful services, a framework called REST2SOAP is proposed. It leverages WADL specification, so it can wrap RESTful services into SOAP services semi-automatically. This framework converts RESTful services into SOAP service. Therefore, it can ignore some features of RESTful services. In order to minimize the interactions between heterogeneous services, the authors of [10] have presented a hybrid approach. Their work contains both a BPEL engine and a REST orchestration engine. Hence, the main workflow is divided into several sub-workflows according to their intrinsic architectural style. Then, each resulted sub-worked is handled in a native way. SOAP-based orchestration is conducted in the BPEL engine, while REST services are orchestrated in the REST orchestration engine. The separation of these two kinds of Web services in a composite service is not a good idea, because it is not applicable in all situations. We found [4] as an outstanding work in the heterogeneous services composition field that enables BPEL to support RESTful services as well as SOAP-based ones. However, it is just an orchestration language like BPEL by the ability of invoking RESTful Web services. Indeed, it does not a comparable method with ours as a composition framework.

The authors of [11] focused on resolving the problem of heterogeneity among service interface and protocols by using Web service adapters. They introduced taxonomy of common mismatches in service interfaces and provided their resolutions.

2.2. Model Driven Service Composition

In the field of model-driven development, the proposed approach in [8] supports dynamic service composition. It uses UML as a method to model service compositions as well as OCL to express business rules, which governs the process of service composition. The authors believe that they can use business rules to determine how a service composition should be structured and scheduled, how the services and their providers should be selected, and how service binding should be conducted. The proposed method uses a model called Information Model (IM) to compose services. Beside these advantages, there are certain shortcomings in this approach that are the lack fulfillment of different service types, service constraints, and evaluation phase.

2.3. Dynamic Service Composition

The [12] is a prototype that guides a user in the dynamic composition of Web services. Authors have developed a semi-automatic process that includes presenting matching services to the user at each step of a composition and filtering the possibilities. The generated composition is then made to execute through WSDL. In [13] the authors have presented a context based Web service composition approaches and provided a comparative study of them, by term context they mean any information that can be used to characterize the entity.

Several approaches have utilized the proxy and adapter

concepts in order to increase the dynamicity and adaptability. In proposed frameworks of [14,15], share the common idea of indirect invocation of Web services to make composite services more adaptable to various runtime changes.

2.4. Summary

Despite the amount of research devoted to Web service composition, very little attention has been paid to the proposing a comprehensive method to take into account some major issues in this field, such as supporting heterogeneous services, being dynamic as well as comprehensive and simple. We believe that MDCHeS framework by introducing three different views of service composition have been able to cover these mentioned issues.

3. The Proposed Framework

A framework is a set of constraints on components and their interaction, and a set of benefits that derive from those constraints. A framework defines a model of computation, which governs the interaction of components [16]. Based on this definition, we proposed MDCHeS framework with three different views: data, process, and component view.

- **Data view** consists of a Meta Model to represent composition elements and their relationship to construct a composite service.
- **Process view** defines the process of designing a composite service that runs through a phased approach. In each phase, a specific model will be derived based on the result model of its previous phase. Indeed, each phase is responsible to convert its input model into a specific output model. The goals of using these models are to increase the abstract, dynamicity, and simplicity level of composition process.
- **Component view** of MDCHeS framework presents components and shows which components interact with each other in order to fulfill the required tasks.

We will illustrate these views in three sub-sections separately in the following.

3.1. MDCHeS Data View

MDCHeS framework performs heterogeneous service composition in a model-driven fashion. Hence, we have designed a *Meta Model*, which is inspired from papers [8,17,18]. Indeed, this *Meta Model* is the schema of MDCHeS repository and has been structured to increase dynamicity and supporting composition of services with different protocols simultaneously. This will happen by supporting the storage of both kinds of Web services in the repository. The elements of our *Meta Model* are *Activity*, *Role*, *Provider*, *Service*, *and Message*. **Figure 1** depicts elements and their relationships. Details of these elements will be as follows:

- **Activity**: This element represents a well-defined concept for a functionally. A composite service is composed of several functionalities. For instance, in a trip planning composite service, functionalities are flight, hotel, and shuttle booking. *Activity* concept is used to have an abstract definition for each group of Web services providing the same functionality. Defining the concept of *Activity* and Service decreases the dependency of composite service to concrete Web

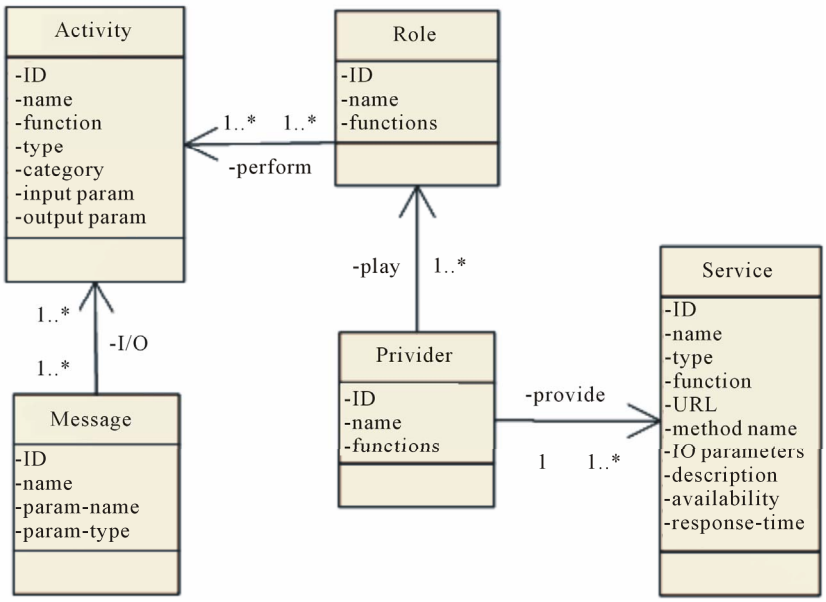

Figure 1. Meta model.

services and leads to a better adaptability and increasing the dynamicity. The main attributes of *Activity* element are *name*, *function*, *type*, *category*, and *input-output parameters*.
  - *Function* describes the main functionality corresponding to *Activity.* It is used as a primary key in relations with other elements.
  - *Type* determines whether the *Activity* is atomic or composite.
  - *Category* is a standard ordination that is used in UDDI (Universal Description Discovery and Integration). For instance, a possible *Category* is "SIC (Standard Industrial Classification)" [19].
  - *Input-output parameters*: these parameters are used to satisfy a specific *Function* of *Activity*. They are in a higher abstraction level than real input-output parameters of concrete Web services.
- **Message**: The input and output parameters of each *Activity* form a *Message*. It includes *name*, *parameters-name* and *parameters-type*.
- **Role**: *Role* is an abstract element that is responsible for performing an *Activity*. Each *Role* contains *name*, and *functions* as its attributes. Each *Role can* do one or more *function*(*s*).
- **Provider**: Each *Provider* provides a concrete service. If a *Provider* supports all functions of a *Role*, it will play that *Role*. The attributes of a *Provider* are *name*, and *functions*.
- **Service**: A *Service* contains all common attributes of a Web service such as *name*, *type*, *function*, *URL*, *method-name*, *input-output parameters*, *description* and some non-functional requirements such as *availability* and *Response-time*.
  - *Type* determines whether the *Service* is SOAP-based or RESTful.
  - *Method-name* in SOAP-based Web service means operations of a service and in RESTful Web service means HTTP method such as GET DELETE, POST, and PUT.
  - *Function*: This is the main functionality that is done by this *Service*. In this paper, it has been supposed that each *Service* just has one *function.* It means that a *Service* with two functions should be saved in the framework Repository as two separated *Services* each with one *function*.

3.2. MDCHeS Process View

As we mentioned before, the proposed approach is based on model-driven principles. In this section, the MDCHeS process view will be discussed in three sub-sections: proposed models to conduct composition process, composition phases and finally proposed composition algorithms, which are used in composition phases.

3.2.1. Proposed Models

MDCHeS utilizes four models to increase simplicity, abstraction and dynamicity of composition process. It is worth mentioning that the goals of these models are not as replacements of some current accepted models such as WSDL, etc. They are introduced to convert the proposed composition process as the transformation of four models based on MDA principles. They are designed according to a *base model* inspired from [20,21] that is depicted in **Figure 2**. The *base model* has four parts as following:

- **Elements (Functionality/Activity/Service/Proxy) definition**: These elements are used in our proposed four models respectively. As the **Figure 2** shows for each element it is needed to introduce several sub-elements. For instance, for *Service* element these things should be mentioned: *service-id*, *provider-id* and *type of service*. These characteristics are extracted from the *Repository* in each phase. The first three elements are described in the previous section; *Proxy* is a Web service that mediates the invocation between a service consumer and a service provider. It decreases the dependency between the composite service and the concrete Web services that are invoked by this composite service. This way, the replacement of alternative concrete services will be possible with a lower cost.
- **Functionality Constraints**: This part is used just for IM (introduced below) to describe user's constraint of each *Functionality* element.
- **Global Constraints**: In this part, the constraints that are related to the through composite service will be introduced. These constraints can be boundary input-output parameters and some non-functional requirements of the composite service.
- **Data & Control Specification**: This part shows Data flow and control flow among elements by using graph notation.

In the following four models that are derived from the introduced *base model* are briefly described.

- **Input-Model (IM)**: It includes a set of requested Functionalities, parameters, constraints and a control flow and a data flow among them. IM also includes non-functional requirements and user's preferences related to a requested *Functionality*. A sample definition for Global constraints in an IM is depicted in **Figure 3**. IM uses all four parts of *base model*. As was mentioned before, *Functionality* is the proper element, which is used in the first part of this model.
- **Abstract Composite Service Model (ACSM)**: If each *Functionality* in an IM is replaced by a proper *Activity*, the result will be an ACSM. Therefore, *Activity* is the used element in the ACSM. Three other parts of IM will be updated based on *Activity* properties in the ACSM.

Functionality/Activity/Service/Proxy definition

Functionality	**Activity**	**Service**	**Proxy**
{name, category};	{activity-id, inputMsg-id, outputMsg-id, role};	{service-id, provider-id, type-of-service};	{proxy-id};

Functionality Constraints

constraint <name>;
input = {/*<name>.input is defined here*/};
output = {/*<name>.output is defined here*/};
QoS = {/*QoS parameters of <name> are defined here*/};

Global Constraints

constraint global <name>, <category name>;

input = {/* boundary input of composite entity are defined here*/};
output = {/* boundary outputs of composite entity are defined here */};
QoS = {/* QoS parameters of composite entity are defined here*/};

Data & Control Specification

/* data and control graph is defined here based on a predefined graph structure*/

Figure 2. Base model.

Global Constraints

//defining boundary inputs, output sand QoS parameters of requested composite functionality.

constraint global Trip-planning, Travel & Tourism;

input = {startDate, finishDate, destination, source, evenName};
output = {planeBooking-id, hotelBooking-id}
QoS = {availability > 70% , responseTime < 0.5 sec}

Figure 3. A sample of IM.GC.

- **Concrete Composite Service Model (CCSM)**: CCSM will be generated by replacing each *Activity* in each ACSM with suitable *Service*(*s*) and related properties. Therefore, used element in this model is *Service*.
- **Executable Proxy-bases Composite Service Model (EPCSM)**: In this model, a Proxy service is generated and used for each *Activity*. The Proxy services are deployed and are made ready to be invoked. After creating an uninitialized EPCSM based on the corresponding generated ACSM, the best CCSMs are used to configure that EPCSM. From user's point of view, this model is a workflow stated in BPEL language that consists of invocation to some Proxy services, which are SOAP-based. Therefore, it can simply be executed by BPEL engine.

From the model-driven point of view, each IM, ACSM, and CCSM is a PIM, because these models are not bounded to a specific platform and are based on some abstract elements such as *Activity*, *Role*, *Provider*, etc and this way they help to have a method with more dynamicity. However, EPCSM is a PSM because it is described by an execution language such as BPEL, so it is a platform specific model.

3.2.2. Composition Phases

Service composition is divided into two main use cases, service composition development and service composition management [1]. In the former, system tries to generate a business process by composing services. This use case starts after receiving a new service request and finally returns an executable composite service, while in the latter, user interacts with the system to execute and manage executable composite services. It begins when the user wants to execute a composite service. This paper focuses on the first use case and supporting the second one has been addressed as a future work for MDCHeS framework.

Since service composition development is a complex process, we utilize a phased approach to compose services. It starts with preparing an abstract specification and gradually makes it more concrete to construct an executable service, based on the given specification. **Figure 4** depicts MDCHeS phases as well as transformation of models.

- **Specification**

Based on [22], there are different levels of user's specification:

  - *Partial specification* that depends on the level of user's domain knowledge and user should outline *Functionalities*.
  - *Full specification* that includes details of all *sub-Functionalities* and their relationship of a composite service.

The goal of this phase is receiving and analyzing the given IM, which clarifies the order and interactions of requested *Functionalities* and their data and control dependencies. Therefore, MDCHeS framework needs Full Specification as its input in the format of IM.

- **Discovery**

In this phase, all *Functionalities* of the received IM should be replaced by available *Activities* that are available at the framework repository. Based on the matching categories of [18], there are three kinds of approaches: keyword matching, parameter matching, and IOPE matching. As this phase is not in our main research focus,

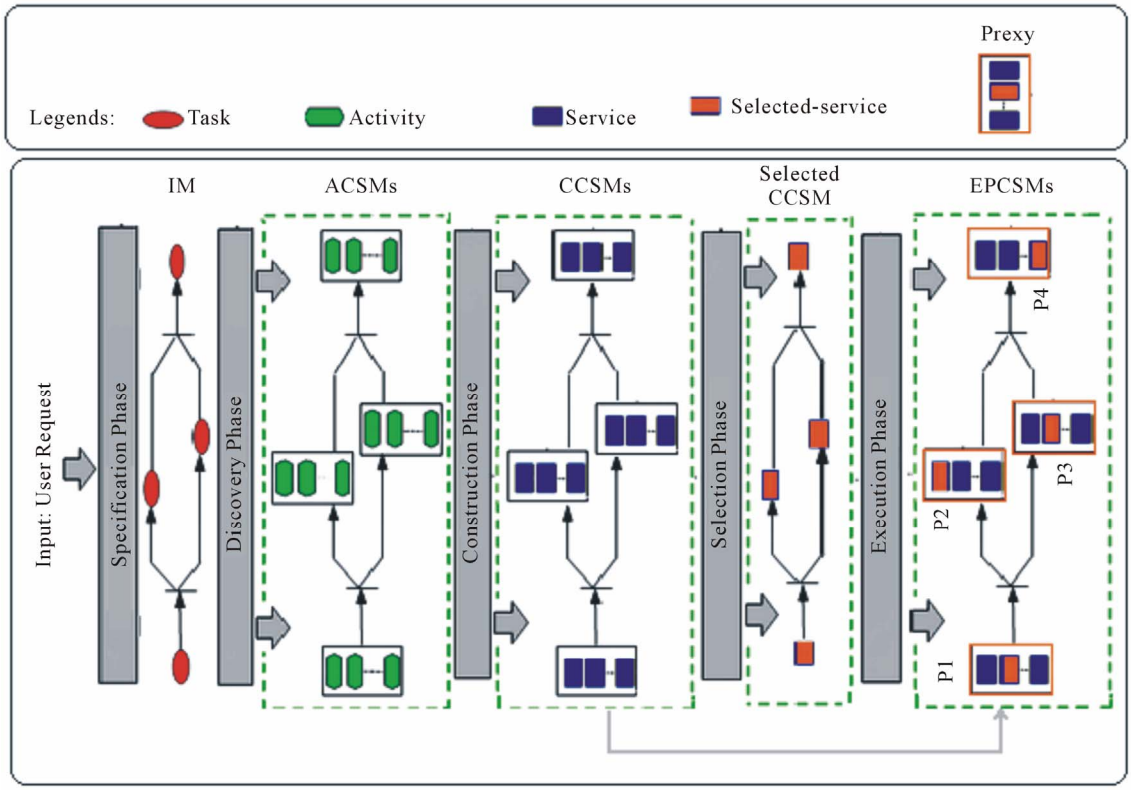

Figure 4. The phases of MDCHeS approach.

we use keyword matching to find common *Function* between *Functionality* and *Activity*, and parameter matching to compare the requested *Functionality* with the existing *Activity* to find the best match of them. We have utilized WordNet [23] to match similar words that using at *Functionalities* and *Activities* as well as their input-output parameters. The *SearchActivity* method (line 31 of **Algorithm 1**), finds and replaces suitable *Activity*(*s*) for each specific *Functionality*. Moreover, if no single *Activity* is found, *ComposeActivities* method (line 45 of **Algorithm 1**) will generate a new composite *Activity* by composing available *Activities* to satisfy user request by a backward method. If this step is unsuccessful, interaction with the user will be necessary.

The output of this phase is ACSM. Since it is possible to find more than one *Activity* for each *Functionality*, a set of ACSMs may be generated for a given IM as the output of **Algorithm 1** (line 30).

- **Construction**

This phase receives a set of ACSM as input and generates related CCSM(s) as output. In this phase, it is time to find appropriate *Roles*, *Providers*, and *Services* for each *Activity*, and then turns ASCM(s) into CCSM(s). Alike the previous phases, it is possible to generate several CCSMs for each ACSM because it is possible to find more than one suitable *Services* for an *Activity*. In order to complete this phase, **Algorithm 2** has been utilized.

- **Selection**

Extracted QoS in *Specification* phase ("Global Constraints" part of IM or IM.GC) and generated CCSM(s) of the previous phase are as two inputs of this phase. For each *Service* at the repository there are some QoS parameters that each provider specifies them. In MDCHeS framework, we consider *Availability* and *Response-time* as QoS of *Service*. Their values are specified in "Global Constraints" part of CCSM (CCSM.GC). Type of *Services* is another factor at this phase, type value is written in CCSM.SD (Service Definition part). Total evaluation factor, which is based on mentioned criteria, is used to order generated CCSM as a ranked set. In order to utilize user's feedbacks and earn more accrue result, this ranked set can be edited via user interactions.

- **Execution**

For producing an executable composite service, one or more ACSMs are received and for each *Activity* inside them, a Proxy service is generated to form an EPCSM. At first, the Proxy services are uninitialized and no concrete *Services* are assigned to them. After the *Selection* phase, a ranked list of concrete *Services* is used to initialize the Proxy services. It means that the high ranked *Services* for each *Activity* are assigned to the corresponding Proxy service. At runtime, the proxy services are responsible for mediating the invocations between the workflow engine, which executes the composite service,

Algorithm 1. Pseudo-code of ACSM generation algorithm.

```
GenerateACSM()
Input: IM;
Output: S_ACCSM as a Set of ACSMs related to given IM;
{
1:    S_Approved = {}
2:    S_NotApproved = {F_i| F_i ∈ IM}
3:    For each Fi in S_NotApproved
4:    S_Fi = SearchActivity (F_i)
//The main loop for making sure there is at least one <Activity_i> for each functionality
5:   While (S_NotApproved is not Empty)
//Trying to fulfill functionalities which there is no single activity for them
6:    For each S_Fi which is empty do
          //Compose some activities as a new ACSM instance to fulfill the requested functionality
7:          S_Fi = ComposeActivities (F_i)
8:                If  S_Fi is NULL then  //If composition fails
9:              interactionResult = InteractWithUser ();
10:             Switch (interactionResult)
11:                   Case "Fi decomposed":
                            //User may decompose the Fi into some more fine-grain Functionalities
                            // to increase the success probability in discovery process
12:                         F_decomposed = all new sub Functionalities for Fi with Related IO(i) and CF
                            //update S_NotApproved
13:                           Remove F_i from S_NotApproved
14:                           Add each F_j ∈ F_decomposed to S_NotApproved
                            //search new functionalities
15:                         For each F_j ∈ F_decomposed
16:                               S_Fj = SearchActivity (F_j)
17:                   Case "Abort":   //User may give up the composition process because he cannot
                            //decompose the functionality anymore
18:                           ShowMessage ("Abort");
19:     For each F_i in S_NotApproved    // taking the user's approval
20:         Similarity_i = ComputeSimilarity (F_i, S_Fi);
            // Removing some improper <activity_i, IO_j> from S_Fi based on the amounts
            // of similarity. It can be done both automatically and by user interactions.
21:         Refine (S_Fi);
22:         If (there is at least one <Activity_i, IO_j> in S_Fi ) then
23:             Add Fi to S_Approved
24:             Remove F_i from S_NotApproved
25:   End While
26:   For each F_i in S_Approved do
27:         For each A_i in S_Fi do
                      //Fetch related Role with the same functionality for each A_i  from the main Repository.
28:                   FetchRole (Ai);
//Making all combinations of different <activity_i, IO_j> ∈ S_Fi for each F_i ∈ IM.FD
//by initializing ACSM.DCG instances from IM.DCG and then replacing the F_i nodes by ctivity_i and related IO and R_i;
//ACSM.AD= for each F_i in IM.FD replace Ai in ACSM.AD and Add R_i;
//ACSM.GC= for each GC.IO in IM.GC replace boundary IO of IM.DFG;
29:   S_ACSM = MakeAllCombinations();
30:   Return S_ACSM;}
-----------------------------------------------------------------------------------------------------------------------------------------------
31: SearchActivity()
      Input: F_i, IM.DF_i
      Output: SActivities as a Set of proper Activities related to given Functionality;
```



```
32:   SActivities = {}
33:   SActivities = FindActivity (Fi..Category);
34:   SfunctionalityNames= FindSimilarWords (Fi.name); // Using WordNet for find all similar words for this Functionality name;
35:   For each Ai in SActivities do
36:         If Ai.name ∈ {SfunctionalityNames} // There is at least one namei in SfunctionalityNames which w is matched with Ai.name)
37:         then
                  //It remains in SActivities
38:               Similarity [Ai] = ComputeIOSimilarity (Ai.IO, Fi.IO);
39:         else
40:               Remove Ai From SActivities
41: For each Ai in SActivities do
42:         If Similarity [Ai] < SimilarityThreshold
43:               Remove Ai From SActivities OR InteractWithUser ()
44: Return SActivities;
```

45: ComposeActivities(F_i)

Input: S_{input}, S_{output};

Output: $S_{globalPossibleGraphs}$;

```
46: SglobalPossibleGraphs = {}
47: N0 = InitilizeNode ();
48: N0.Input = Soutput;
49: G0 = InitializeGraph ();
50: G0.Root = N0;
51: SglobalPossibleGraphs = GenerateCompositionGraph (G0, Sinput);
52: Return SglobalPossibleGraphs;
```

53: GenerateCompositionGraph()

Input: initialGraph, S_{input};

Output: $S_{localPossibleGraphs}$;

```
54: Sleaf = GetLeafs (initialGraph);
55: SlocalPossibleGraphs = {}
56: For each leafi ∈ Sleaf do
57:   For each pi ∈ leafi.Input do
58:         If (pi ∈ Sinput)
            // Input parameter is satisfiable and can be removed from the list
59:               Remove pi From Sinput
60:               MarkAsBoundryInput (pi);
61:               If(Sinput == {})
62:                     return initialGraph;
63:               else
64:                     SActivity = FindActivityByOutput (pi);
65:                     If(SActivity.size == 0)
66:                           Return {}; //Fail
67:                     firstActivity = SActivity.begin ();
68:                     initialGraph.AddChild (firstActivity, leafi, pi);
69:                     Remove firstActivity from SActivity
70:                     If(SActivity.size != 0) //If there is more activity
71:                           For each activityi ∈ SActivity do
72:                                 newGraph = InitiateFromExistingGraph (initialGraph);
73:                                 newGraph.AddChild (activityi, leafi, pi);
74:                                 SresultedGraph= GenerateCompositionGraph (newGraph, Sinput);
75:                                 SlocalPossibleGraphs = SlocalPossibleGraphs ∪ SresultedGraph;
76:                                 SresultedGraph= GenerateCompositionGraph (initialGraph, Sinput);
77: SlocalPossibleGraphs = SlocalPossibleGraphs ∪ SresultedGraph;
78: Return SlocalPossibleGraphs;
```


Algorithm 2. Pseudo-code of EPCSM generation algorithm.

GenerateCCSM ()

Input: S_{ACSM} as a Set of ACSMs, and IM for constraints;

Result: S_{ccsm} as a Set of CCSMs related to given S_{ACSM};

1: $S_{CCSM} = \{\}$; //A Set of concrete models which is initially empty

2: **For** *each* $acsm_i.DCG$ *that* $acsm_i \in S_{ACSM}$ **do**

// DCG means Data Control Graph that is in Data & Control Specification part of proposed four models.

3: Create $ccsm_j$ *as a CCSM instance and initialize it from the* $acsm_i$;

4: **For** *each nodes and related edges* $\in acsm_i.DCG$ **do**

//Finding proper providers and services for this given activity

5: $Services_i = \underline{FindServices}\ (activity_i, IO_j, IM.FC_j)$;

// Discover proper services based on the predefined relationship between message, activity, role, provider and service in the Main Repository

Updating the ccsmj by these discovered services.

// The corresponding providers are added implicitly.

6: Add $Services_i$ *and* $Provider_i$ *to the* $ccsm_j.SD$ *and* $ccsm_j.DCG$;

7: Compute *totsl QoS for each* $ccsm_i$ *based on the QoS of each contained Services and complete* $ccsm_i.GC$;

8: Add *type of Services to* $ccsm_i.DS$;

9: Add $ccsm_j$ to $S_{CCSM;}$

10: **Return** $S_{CCSM;}$

//A set of ranked CCSMs will be generated among all ccsm instances which stored in S_{CCSM} based on the evaluation factor

//In order to take user confirmation, user's interactions can be made at this time.

11: $OrderedS_{CCSM} = RankedCCSMs\ (S_{CCSM}, IM..GC)$;

// In this step for each Activity of ACCSM, a proxy will be initialized based on the best proper service for that Activity.

12: $EPCSM = \underline{MakeExecutable}\ (theFirstRankedCCSM, ACCSMs)$;

13: **Return** S_{EPCSM};

and concrete services. Beside the mediation, they are responsible for resolving the interface incompatibilities. We all know that the services, even with the same functionality, may have different interfaces. These differences can be in type, order or the number of input-output parameters. The concrete services can also support different protocols. The Proxy services are used to resolve these incompatibilities through a process called Interface Mapping. For example, when a Proxy service receives a SOAP-based request but the goal concrete service is RESTful, the proxy service maps the incoming request to a RESTful invocation, which the selected Web service expects.

3.2.3. Proposed Algorithms

MDCHeS composition process is the transformation of four models to each other that happens in each phase. This transformation is done by two main algorithms that should be run sequentially. (**Algorithms 1** and **2**). The first algorithm produces a composite service in the form of an abstract model and then the second algorithm generates a proxy-based executable composite service. In the following, pseudo-codes of these algorithms have been shown. They are applied on a case study in section 0 to illustrate how they work.

3.3. MDCHeS Component View

In two previous sub-sections, we elaborated data and process view of our framework. In this part, component view will be introduced. **Figure 5** depicts MDCHeS framework architecture. In the following, we state a brief description of each component:

- **Request Analyzer**: This component receives the user's request, which is in IM format, as input, and then analyzes the given IM and sends each part of it to the suitable components.
- **Main-Repository**: All available composition elements such as *Activity*, *Message*, *Role*, *Provider*, and *Service* as well as some needed elements to cover their relationships will be stored in the *Main-Repository* in order to fulfill the user requirements. The schema of this repository is derived from *Meta Model*, which was illustrated in **Figure 1**.
- **Discovery-Engine**: This component is responsible for communication with *Main-Repository* to find proper elements for each component. This component uses keyword and parameter matching for its search as well as WordNet [23]. As we can see at **Figure 5**, if each component needs the interaction with *Main-Repository*, it will do it via the mediation of this component. Indeed, it has following responsibilities:

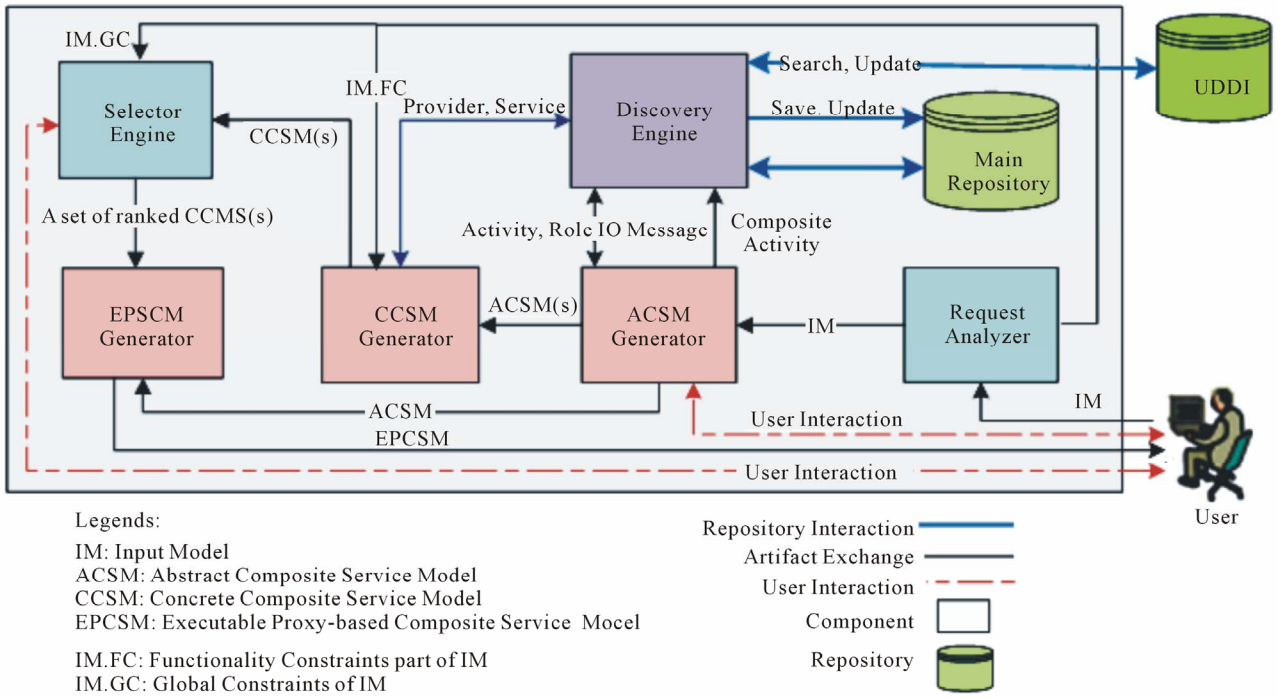

Figure 5. MDCHeS framework.

- Searching suitable Activity, Role and IO-Message elements for *ACSM-Generator* component based on the given *Functionalities* of each IM in the *Discovery phase.* In this phase, if our method generates a *Composite Activity*, the *Activity* will be stored in the *Main-Repository* for further reuse.
- Searching proper *Provider* and *Service* elements based on the related *Activities* of each ACSM for *CCSM-Generator* component at the *Construction* phase.
- Updating Main-Repository elements via communicating with UDDI.

- **ACSM-Generator**: This component generates all possible ACSMs by replacing each *Functionality* with its relevant discovered *Activity*(s) set. *ACSM-Generator* uses *Discovery-Engine* component to search proper elements of the *Main-Repository*.
- **CCSM-Generator**: As mentioned before, our composition approach is an incremental process. Hence, requested composite service will be completed in each step gradually. In order to create CCSM, this component receives ACSM(s) and transforms it/them into a CCSM by replacing the proper *Provider* and *Service* elements for each *Activity* by interacting with *Discovery-Engine.* Furthermore, some relevant user's preferences are applied for generating CCSMs. These preferences, as **Figure 5** shows, have been taken from "Functionality Constraints" part of IM (IM. FC), which will be used at this component as an input. For instance, if using of a specific *Provider* was recommended by user, this component would only select the provided *Services* of the mentioned *Provider*.
- **Selector Engine**: This component receives the output of the *Request Analyzer* component that contains user preferences from "Global Constraints" part of IM (IM. GC), and CCSM(s) of CCSM-Generator component. It performs the *Selection phase* and provides a set of ranked CCSMs based on the best match to evaluation factor, which was introduced at the *Selection phase*
- **EPCSM-Generator**: This component is responsible to generate the output, executable composite service model. It receives one or more ACSM(s) and generates an EPCSM by replacing each *Activity* with a Proxy service. Later these proxy services will be initialized and configured by a set of ranked CCSMs produced by the *Selector Engine.*

4. Evaluation

In this section, in order to show the applicability, and feasibility of mentioned algorithms, a case study is applied to them. After that, capabilities of our framework are evaluated through a statistical analysis of experts' answers to the questions shown in **Table 1**. Finally, some main factors to evaluate a composition framework are mentioned and then our proposed framework is compared with some leading frameworks.

4.1. Case Study

We applied the proposed algorithms on a real world

Table 1. Experts' evaluation of the MDCHeS framework.

Scope of capabilities	MDCHeS capabilities	1	2	3	4	5	np	m	sd
Input	It is not need that user who requests a composite service has a deep technical knowledge.	0	0	4	5	2	7	3.81	0.75
	MDCHeS consider both functional and non-functional requirements in constructing requested composite services.	0	4	2	3	2	5	3.27	1.19
	MDCHeS uses a suitable format to take user's requirements.	0	1	5	2	2	4	3.50	0.86
Process	This method has the ability to use organization assets and reuse existing composite services.	0	0	0	8	3	11	4.27	0.46
	MDCHeS can use all types of Web services in outside of the organization to use in composite services.	0	0	1	5	5	10	4.36	0.70
	Using MDCHeS framework facilitates designing of composite services.	0	0	1	8	2	10	4.09	0.47
	MDCHeS is practical in the scope of an enterprise.	0	3	5	3	0	3	3.00	0.77
	MDCHeS framework can be adapted to dynamic Web service environments.	0	2	5	4	0	4	3.18	0.75
	By using MDCHeS framework more and more in an enterprise, the successful response rate to user requests will increase.	0	0	0	3	7	10	4.70	0.48
Output	MDCHeS output (EPCSM) is a suitable model to cover dynamicity in the ntime.	0	0	1	6	4	10	4.27	0.64
	EPCSM uses proper techniques to use heterogeneous services in a composite service.	0	1	0	8	2	10	4.00	0.77

scenario which is inspired from [24,25]. As a prerequisite, we prepared a data set to simulate MDCHeS *Main- Repository* elements based on the structure of *Meta Model* by using Microsoft Access as DBMS. To save space, only a few rows of each table of this data set are shown in **Figure 6**. We run our case study on this data set. The scenario is as follows: "A user wants to arrange his travel in order to attend to an event such as: a conference or a business meeting. Therefore, he must be able to register to that event and arrange for accommodation as well as his transportation. Moreover, he wants to get the weather forecast of the destination city at that period of time."

The MDCHeS framework can realize the specified scenario by using the available elements at the *Main-Repository* and applying the proposed algorithms on them. As we mentioned before, **Figures 6(a)-(i)** depict a small snapshot of available *Activity*, *Message*, *Role*, *Provides*, *Service* elements and some other tables which cover the relationships among these elements.

Based on the input of **Algorithm 1**, **Figure 7** is accepted as IM; it is worth mentioning that, we just show the graphs of "Data-Control Specification" part (DCG) of each model as generated artifacts, because of the space limitation. Furthermore, for the sake of simplicity, we have not shown inputs and outputs of each graph.

At first, each *Functionality* of **Figure 7** will be replaced by its relevant <*Activity*, *Role*, *IO-Message*> tuple by invoking *SearchActivity* (*Fi*) function (line 4 of **Algorithm 1**). This function (line 31 of **Algorithm 1**) finds one or more appropriate *tuple*(*s*) for *Flight*, *Event*, *Hotel-Shuttle* and *car* functionalities. The nodes of **Figures 9(a)** and **(b)** show found tuple.

In this example, there is no suitable found pair for *Weather Functionality* with city-name as its input at the *Main-Repository* (line 6 of **Algorithm 1**). Hence, *ComposeActivities* (*Fi*) function (line 7 of **Algorithm 1**) will be called. This function (line 45 of **Algorithm 1**) aims to make a proper composite *Activity* for this *Functionality* by composing available atomic *Activities*. This function firstly scans Tables 1(d), (f), (h), and (a) to find suitable *Activity* with the same output messages as output messages of the *Weather*. The <*A*7, *R*7, *M*10, *M*25> set will be selected in this step. Then, it tries to find appropriate <*Activity*, *Role*, *IO-Message*> tuple with the same output message set as input message set of the selected pair. This step will be performed repeatedly until appropriate pair would be constructed to satisfy *Weather Functionality*. The result for this *Functionality* would be the composition of <*A*7, *R*7, *M*10, *M*25> and <*A*8, *R*8, *M*11, *M*26> as **Figures 8(a)** and **(b)** show.

In the next steps, **Algorithm 1** examines each <*Activity*, *Role*, *IO-Message*> to be decided as being "Approved" or "Not approved", when the algorithm can find at least one approved set for each *Functionality*, it marks the *Functionality* as "Approved" (line 19 of **Algorithm 1**). If it finds no suitable *Activity* for at least one given *Func-*

Activity (a)

ID	name	function	category	role-ID
A-01	HotelReservation	Hotelbooking	Hotels	R-04
A-02	EventRegistration	EventRegisterin	Travel & Tourism	R-01
A-03	PlaneReservation	PlaneBooking	Airlines	R-02
A-04	CarReservation	CarRenting	Automobiles & Parts	R-06
A-05	ShuttleReservation	ShuttleBooking	Hotels	R-05

Role (c)

ID	name
R-01	EventRegisterationAgent
R-02	Airline
R-04	HotelAgent
R-05	ShuttleAgent
R-06	CarRentingAgent

Provider (e)

ID	name
P-01	MahanAir
P-02	AsemanAir
P-03	GoldCar
P-04	LuxeCar
P-05	GoTrip

Service (b)

ID	name	tyoe	URL	method name	availability (%)	response time (ms)	input parameters	output parameters	Activity ID	Provider ID	IO ID
S-01	MahanAirB ooking	SOAP-based	http://	Booking	91	600	String from, String to, Date departDate, Date returnDate;	Num seatNo, Num flightNo, String flightBooking_ID	A-03	P-01	IO-06
S-02	AsemanAir Booking	SOAP-based	http://	Booking	97	100	String origination, String to, Date departeDate, Number passenger	Num seatNo, Num flightNo, String flightBooking_ID	A-03	P-01	IO-05
S-03	Gold CarRenting	RESTful	http://	POST	98	200	String pickupCity, Date pickupDate, String	String carBooking_ID, String pickupAddress	A-04	P-03	IO-07
S-04	LuxeCar Renting	RESTful	http://	POST	99	400	string depature_address, Date date_leave, String	String carBooking_ID, String pickupAddress	A-04	P-04	IO-08
S-17	MDCHeS-WeatherForcasting	RESTful	http://	GET	96	500	String cityName	String forcasted WeatherOfCity	A-13	P-14	IO-13

Message (d)

ID	name	Parameters
M-01	HotelBookingInputMsg	String hotelName; Date dateArrive; Date dateBack
M-02	EventRegisteringInputMsg	String event_name, Date start_registr_date
M-03	EventRegisteringInputMsg2	String event_name, Date start_registr_date, String duration
M-04	PlaneBookingInputMsg	String origination, String to, Date departeDate, Number passenger
M-05	PlaneBookingInputMsg2	String from, String to, Date departDate, Date returnDate; Boolean journey type;

Role-Activity (i)

Activity-ID	Role-ID
A-01	R-04
A-02	R-01
A-03	R-02
A-04	R-06
A-05	R-05

Role-Provider (g)

Provider-Id	Role-ID
P-01	R-02
P-02	R-02
P-03	R-06
P-04	R-06
P-05	R-04

Activity-Input.Output (h)

Activity-ID	IO-ID
A-01	IO-01
A-01	IO-02
A-02	IO-03
A-02	IO-04
A-03	IO-05

Input-Output (f)

IO-ID	Input-Message ID	Output Message ID
IO-01	M-01	M-19
IO-02	M-13	M-19
IO-03	M-02	M-20
IO-04	M-03	M-20
IO-05	M-04	M-21

Figure 6. (a)-(i) A brief snapshot of main-repository elements.

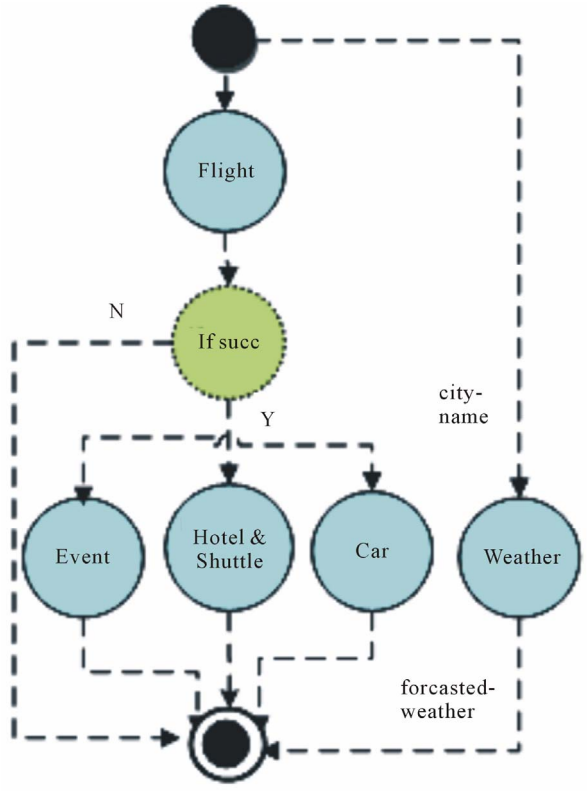

Figure 7. IM.

tionality (line 8 of **Algorithm 1**), the algorithm will interact with user to decompose given *Functionalities* and will repeat mentioned steps for them again until it achieves a successful result.

As the last step, different ACSMs will be generated based on <*Activity*, *Role*, *IO-Message*> combinations. The constructed ACSMs of the IM (**Figure 7**) are shown in **Figures 9(a)** and **(b)**. In this step two tasks should be done.

In one hand, one or more CCSM(s) should be generated. For this aim, **Algorithm 2** receives the outputs of **Algorithm 1** as well as IM.FC (line 7 of **Algorithm 2**). Then, <*Provider*, *Service*> tuple will be extracted for each <*Activity*, *Role*, *IO-Message*> tuple and forms the CCSM(s) represents the resulted CCSMs for the given ACSM of **Figure 9(a)**.

On the other hand, ACSMs are used to produce an uninitialized EPCSM. It will be initialized after determining a set of ranked CCSM(s), which is finalized via user interaction. User can select one of the generated CCSMs (without user interaction the highest ranked CCSM will be sent as output).

In addition, **Algorithm 2** selects two possible <*Provider*, *Service*> tuple for some nodes of ACSM in **Figure 9(a)**. Hence, there are two versions of CCSMs, which are represented in **Figures 10(a)** and **(b)** as two results of this step. Finally, the algorithm fills Global Constraint

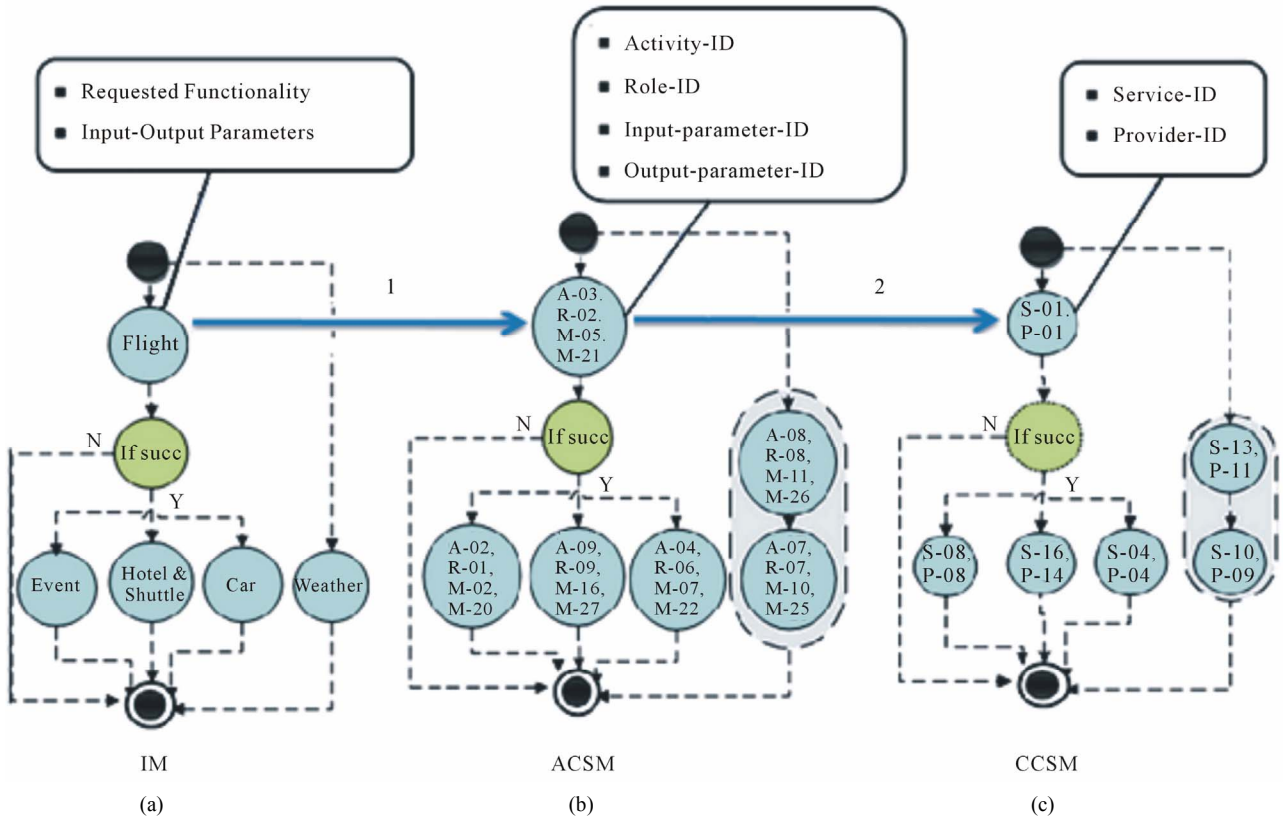

Figure 8. (a)-(c) Model transformation of "Trip Planning" scenario.

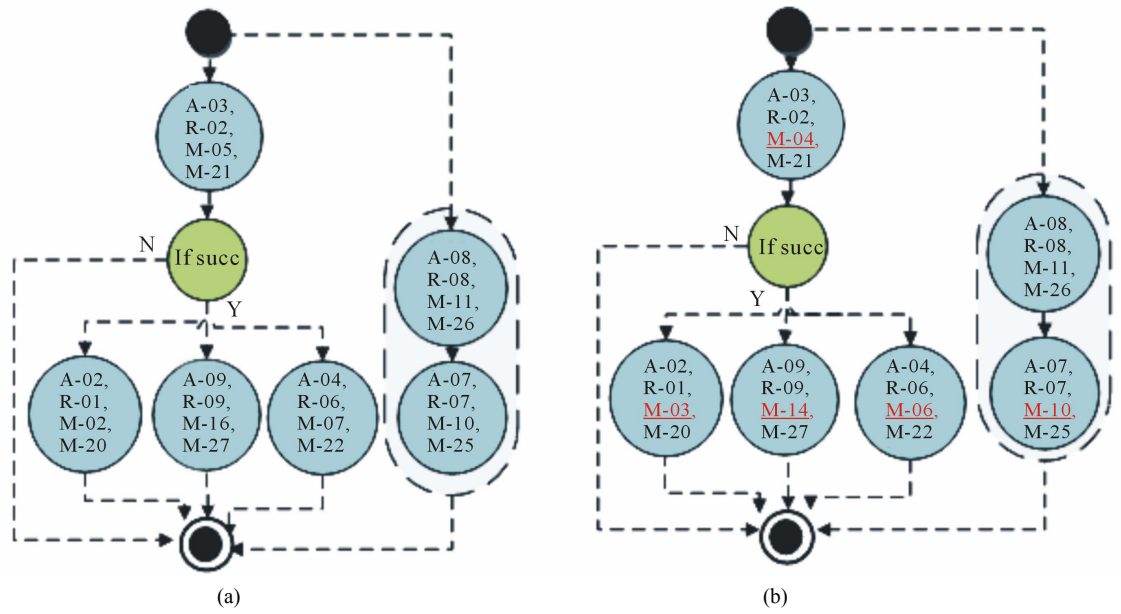

Figure 9. (a), (b) Generated ACSMs.

(CCSM.GC) and Service Definition (CCSM.SD) parts of these two CCSMs and rank them based on evaluation factor (line 7 of **Algorithm 2**).

In this step, we have the most proper CCSM, which is **Figure 10(a)** in this scenario, so the Proxy services inside the EPCSM (nodes of **Figure 11**) can be initialized by it (line 11 of **Algorithm 2**). The EPCSM that is depicted in **Figure 11** is completed and is ready to be delivered to user for further invocations. In **Figure 11**, the colored services are the initiated ones that have been had

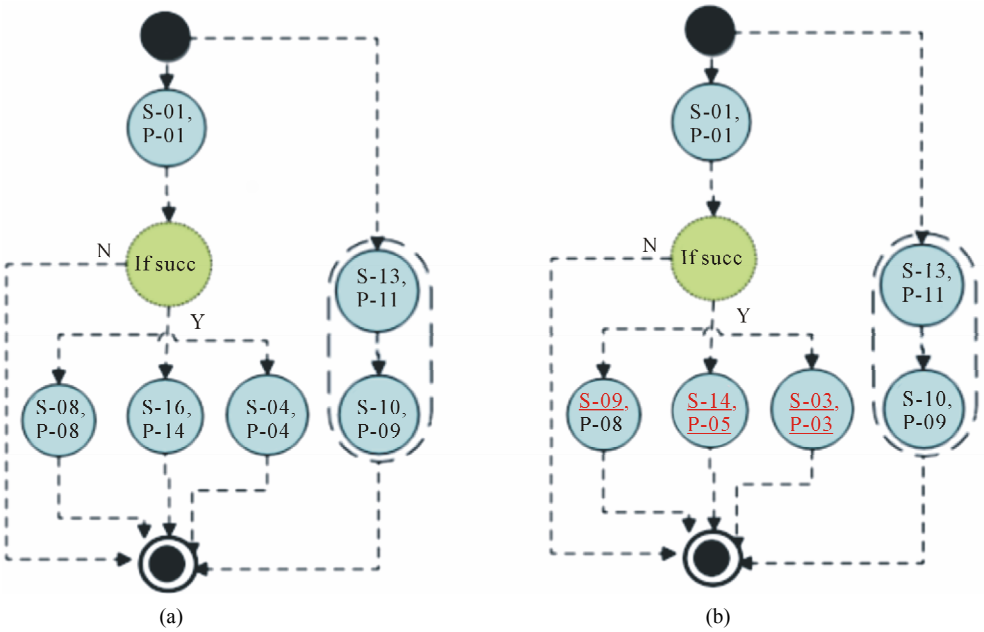

Figure 10. (a), (b) Generated CCSMs.

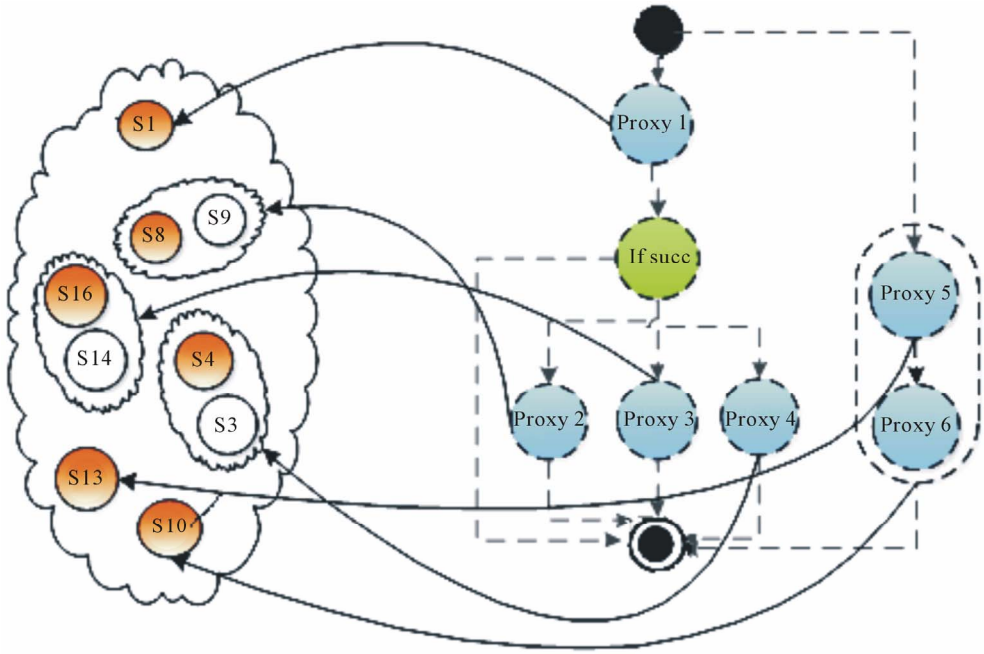

Figure 11. EPCSM.

selected based on the nodes of CCSM of **Figure 10(a)**. In **Figure 8**, we can see the transformation of IM to ACSM and then CCSM, which is done by **Algorithms 1** and **2**.

4.2. Experts' Evaluation of the MDCHeS Framework

The experts' evaluation of MDCHeS was collected through a survey as a workshop at ASER research group. We had eleven participants in our survey and all of them had rich experience in software development as well as suitable knowledge in service-oriented architecture. The questions had been organized into three categories regarding to inputs, outputs, and our composition process. The participants could answer a question based on a five-point scale ranging from 1) strongly disagree, 2) disagree, 3) neutral, 4) agree, to 5) strongly agree.

In **Table 1**, np expresses the number of positive responses, m represents average of the given grades, and sd

denotes the standard deviation.

4.3. MDCHeS Comparison

In order to compare the capabilities of MDCHeS framework with capabilities of some current leading frameworks, a set of criteria are needed. Since, different authors have proposed various criteria for this goal; we chose a common set of them. They are introduced briefly in the following. Then, we have compared it among current frameworks in **Table 2**. Before this comparison, these frameworks will be introduced.

4.3.1. Composition Factors

- **Composition Strategy** [26]

Existing composition strategies in the field of service composition are model driven, semantic, formal, dynamic, and aspect-oriented. We utilized model-driven as MDCHeS composition strategy.

- **Automation Level** [5,27]

The process of describing user's requested workflow, selecting suitable services to compose, and composing selected services that can be performed in the following categories:

- *Automatically*, *i.e.*, using a tool that implements a composition algorithm.
- *Semi-automatically*, *i.e.*, in case that a user makes choices during the composition phase aided by an interactive tool.
- *Manually* that will be done by user.

As mentioned before, the proposed framework performs service composition in a semi-automatic fashion. Although the major parts of development can be done automatically, some interactions can be made to take the user confirmations in *Discovery*, *Construction*, and *Selection* phases. We believe methods in this category work better than others do, because user's feedbacks can lead to a more satisfactory composite service.

- **Composition logic** [5]

Composition logic is defined as how the interactions among participating services take place in the composition process. Accordingly, composition logic has been presented in the three categories: process-driven, rule-driven, and hybrid.

In the process-driven mechanism, the composition uses process definitions, which business logic is expressed as an input model to specify possible interactions among the participating Web services. In rule-based, business rules use to specify composition logic. In hybrid, business logic divides to business processes that are made by business rules. The composition logic of MDCHeS is process-driven.

- **Composition time** [28]

It refers to the moment at which the approaches perform the service composition synthesis. Two distinct moments may exist: design-time, and runtime. MDCHeS create composite service at design time.

- **Support heterogeneous services** [9,28,29]

Heterogeneous in this concept means in the level of technology and using protocols. Therefore, there are two types of Web service, SOAP-based and RESTful.

4.3.2. Introduce Leading Composition Frameworks

In the following, a brief description of some current leading composition framework is presented. The capabilities of these frameworks compare in **Table 2** with the features of MDCHeS.

eFlow [30,31] is a framework to support specify, enactment, monitor and run Web services. Composition had been done by graph theory and composite services have been modeled as business processes. This is an industrial method now.

METEOR-S project [32] was established as a dynamic framework to compose Web services. It focuses on design time composition by developing patterns that are able to bind dynamically in the run-time.

WebTransact approach [33] includes a layered framework to provide a structure to construct reliable composite service. It uses WSDL to describe service functionality and WSTL (Web Services Transaction Language) to facilitate Web service functionalities.

In [34] a framework called **DynamiCoS** is presented. It supports different phases to provide automatic service discovery, selection, and composition process. To achieve this automated support, DynamiCoS utilizes ontology.

Table 2. MDCHeS framework comparison.

Composition framework	Composition Strategy	Automation level	Composition logic	Composition time	Support heterogeneous services
MDCHeS	Model driven	Semi-automate	Process Oriented	Design	Yes
eFlow [30,31]	Dynamic	Semi-automate, Manual	Process Oriented	Design	No
METEOR-S [32]	Semantic	Semi-automate, Manual	Rule Oriented	Design	No
WebTransact [33]	Dynamic	Semi-automate, Manual	-	-	No
DynamiCoS [34]	Semantic	Automate	Process Oriented	Runtime	No

As **Table 2** shows, although MDCHeS is a semi automated framework, it has the capability of supporting heterogeneous Web services by using Model Driven princeples that have facilitated its composition process.

5. Conclusion and Future Work

In this paper, MDCHeS framework was proposed to compose heterogeneous Web services dynamically. We introduced this framework via three different views: data, process, and component view. Each view provides certain instructions to cover the whole process of composing Web services. In data view, the *Meta Model* embraces all required elements to handle composition of both both SOAP-based and RESTful Web services.

In process model, phased approach fulfills user's requests in a piecemeal fashion through five phases: Specification, Discovery, Construction, Selection, and Execution. In addition to apply proposed algorithms on MDCHeS models. In component view, introduced components as well as their responsibilities and interactions provide software architecture to fulfill designing Web service composition.

MDCHeS framework receives IM as the input model and eventually generates an executable proxy-based composite service semi-automatically as the output model.

In order to cope with the complexity of the composition process, model-driven principles have been used in MDCHeS framework.

Although MDCHeS framework is the enhanced version of [35], we have defined some future work to enhance it more.

Supporting "service composition management", which was described before, can be a significant future work of this research. Using Activity, Role, Provider and Proxy concepts and defining ACSM enables our method to generate the best composite service dynamically in cases that the underlying layer (physical services) has been changed. However, there are some other aspects of dynamicity, which are relevant to dynamic replacement or reconfiguration of services at runtime; they will be addressed as the future work. Supporting of more QoS parameters and apply them on Discovery and Selection phases is yet another future work.

6. Acknowledgements

This project was partially funded by Shahid Beheshti University under the program Automated Software Engineering Research Group, Faculty of Electrical and Computer Engineering.

REFERENCES

[1] C. Pautasso, "On Composing RESTful Services," *Proceedings of IEEE Internet Computing*, Vol. 12, No. 5, 2008, pp. 24-31.

[2] B. Benatallah, R. Dijkman, M. Dumas and Z. Maamar, "Service Composition: Concepts, Techniques, Tools and Trends," In: Z. Stojanović and A. Dahanayake, Eds., *Service-Oriented Software System Engineering: Challenges and Practices*, Idea Group, 2005, pp. 48-66.

[3] C. Pautasso, "Composing Restful Services with JOpera," *Proceeding of the International Conference on Software Composition* (SC09), Zurich, July 2009, pp. 142-159.

[4] C. Pautasso, "RESTful Web Service Composition with BPEL for REST," *Data & Knowledge Engineering Journal*, Vol. 68, No. 9, 2009, pp. 851-866. doi:10.1016/j.datak.2009.02.016

[5] C. Pautasso, "BPEL for REST," Business Process Management, Springer, 2008, pp. 278-293. doi:10.1007/978-3-540-85758-7_21

[6] F. Lécué, E. Silva and L. F. Pires, "A Framework for Dynamic Web Services Composition," *Proceedings of the 2nd ECOWS Workshop on Emerging Web Services Technology* (WEWST07), Halle, 26 November 2007.

[7] R. Khadka and B. Sapkota, "An Evaluation of Dynamic Web Service Composition Approaches," *Proceedings of the 4th International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing* (ACT4SOC), Athens, 23 July 2010, pp. 67-79.

[8] B. Orriens, J. Yang and M. Papazoglou, "Model Driven Service Composition," *Proceedings of Service-Oriented Computing Conference* (ICSOC), Trento, 15-18 December, 2003, pp. 75-90.

[9] Y. Peng, M. Shang-Pin and L. Jonathan, "REST2SOAP: A Framework to Integrate SOAP Services and RESTful," *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications* (SOCA' 09), Taipei, 14-15 December 2009, pp. 106-109.

[10] K. He, "Integration and Orchestration of Heterogeneous Services," *Proceedings of the IEEE Joint Conferences Pervasive Computing* (JCPC), 3-5 December 2009, pp. 467-470.

[11] W. Kongdenfha, H. R. Motahari-Nezhad, B. Benatallah, F. Casati and R. Saint-Paul, "Mismatch Patterns and Adaptation Aspects: A Foundation for Rapid Development of Web Service Adapters," *IEEE Transactions on Services Computing*, Vol. 2, No. 2, 2009, pp. 94-107. doi:10.1109/TSC.2009.12

[12] K. Fujii and T. Suda, "Dynamic Service Composition Using Semantic Information," *Proceedings of the 2nd International Conference on Service Oriented Computing* (ICSOC'04), November 2004, pp. 39-48.

[13] Y. Jadeja and K. Modi, "Context Based Dynamic Web Services Composition Approaches: A Comparative Study," *International Journal of Information and Education Technology*, Vol. 2, No. 2, 2012, pp. 164-166.

[14] M. Di Penta, R. Esposito, M. Villani, R. Codato and M. Colombo, "WS Binder: A Framework to Enable Dynamic Binding of Composite Web Services," *Proceedings of the International Workshop on Service-Oriented Software Engineering*, ACM, Shanghai, 2006, pp. 74-80.

[15] A. Erradi and P. Maheshwari, "Dynamic Binding Framework for Adaptive Web Services," *Proceedings of the* 3*rd International Conference on Internet and Web Applications and Services*, 8-13 June 2008, pp. 162-167. doi:10.1109/ICIW.2008.121

[16] E. Lee, "What's Ahead for Embedded Software," *Proceeding of the IEEE Computer*, Vol. 33, No. 9, 2000, pp. 18-26.

[17] W. Junye, M. Lirui and C. Hongming, "A REST-Based Approach to Integrate Enterprise Resources," *Proceedings of the International Forum on Computer Science-Technology and Application*, Vol. 3, 2009, pp. 219-223.

[18] L. Liu, Z. Wu, Z. Ma and W. Wei, "Functionality Semantic Indexing and Matching Method for RESTful Web Services Based on Resource State Descriptions," *Proceeding of the World Congress on Computer Science and Engineering* (*WCSE*), Xiamen, 19-21 May 2009, pp. 138-142.

[19] Wikipedia, "Standard Industrial Classification," 2012. http://en.wikipedia.org/wiki/Standard_Industrial_Classification

[20] F. Rosenberg, P. Leitner, A. Michlmayr, P. Celikovic and S. Dustdar, "Towards Composition as a Service—A Quality of Service Driven Approach," *Proceeding of the IEEE* 25*th International Conference on Data Engineering*, 29 March-2 April 2009, pp. 1733-1740. doi:10.1109/ICDE.2009.153

[21] F. Rosenberg, "QoS-Aware Composition of Adaptive Service-Oriented Systems," Ph.D. Dissertation, TU Vienna University, Vienna, 2009.

[22] A. Kim, M. Kang, C. Meadows, E. Ioup and J. Sample, "A Framework for Automatic Web Service Composition," Technical Report of Naval Research Lab Washington DC, 2009. http://handle.dtic.mil/100.2/ADA499917

[23] Wikipedia, "WordNet," 2012. http://en.wikipedia.org/wiki/WordNet

[24] R. Hull, M. Hill and D. Berardi, "Semantic Web Services Usage Scenario: e-Service Composition in a Behavior Based Framework," Semantic Web Services Initiative Language, 2005.

[25] T. De Giorgio, G. Ripa and M. Zuccala, "An Approach to Enable Replacement of SOAP Services and REST Services in Lightweight Processes," *Proceedings of the* 10*th International Conference on Current Trends in Web Engineering*, Vienna, July 2010, pp. 338-346.

[26] S. Dustdar and W. Schreiner, "A Survey on Web Services composition," *International Journal of Web and Grid Services*, Vol. 1, No. 1, 2005, pp. 1-30. doi:10.1504/IJWGS.2005.007545

[27] R. Pessoa, E. Silva, M. Sinderen, D. Quartel and L. Pires, "Enterprise Interoperability with SOA: A Survey of Service Composition Approaches," *Proceeding of the* 12*th IEE Enterprise Distributed Object Computing Conference*, Auckland, 1-4 September 2009, pp. 238-251.

[28] H. Zhao and P. Doshi, "Towards Automated RESTful Web Service Composition," *Proceeding of IEEE International Conference on Web Services*, 6-10 July 2009, pp. 189-196. doi:10.1109/ICWS.2009.111

[29] Z. Haibo, "Scalable Composition of Web Services Under Uncertainty," Ph.D. Dissertation, University of Georgia, Vienna, 2009.

[30] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy and M. Shan, "eFlow: A Platform for Developing and Managing Composite e-Services," *Proceeding of IEEE Academia/ Industry Working Conference*, New York, 27-29 April 2000, pp. 341-348.

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

[32] R. Aggarwal, K. Verma, J. Miller and J. Milnor, "Dynamic Web Service Composition in METEOR-S," *Proceeding of the World Wide Web Conference*, Athens, 2004.

[33] P. Pires, "WEBTRANSACT: A Framework for Specifying and Coordinating Reliable Web Services Compositions," Technical Report ES-578/02, Federal University of Rio De Janerio, 2002. http://www.cos.ufrj.br/~pires/webTransact.pdf

[34] E. Silva, F. Pires and M. Sinderen, "Supporting Dynamic Service Composition at Runtime Based on End-User Requirements," *Proceedings of* 1*st International Workshop on User-Generated Services*, Stockholm, 24 November 2009.

[35] S. Farokhi, A. Ghaffari, A. Nikravesh and F. Shams, "A Model Driven Framework to Compose Heterogeneous Services," *Proceeding of the International Conference on Information Science and Applications* (ICISA'11), Jeju-do, 2011.

