WEB SERVICE COMPOSITION BASED ON POPULARITY

Selwa Elfirdoussi[1], Zahi Jarir[1], Mohamed QUAFAFOU[2]

[1]Laboratory LISI, Computer Science Department, Faculty of Sciences,
Cadi Ayyad University, BP 2390, Marrakech, Morocco
s.elfirdoussi@ced.uca.ma ; jarir@uca.ma
[2]LSIS – UMR CNRS 6168, Domaine universitaire de St Jérôme,
F-13397, Marseille Cedex 20, France
mohamed.quafafou@univ-amu.fr

ABSTRACT

In Web Service research, providing methods and tools to cater for automatic composition of services on the Web is still the object of ongoing research activity. Despite the proposed approaches this issue remains open. In this paper we propose a seamless way to compose automatically web services from expressed abstract process model. The process of composition is based on web service popularity concept. To validate our approach an implementation is presented.

KEYWORDS

Automatic Web service composition, Web service selection, Web service popularity.

1. INTRODUCTION

Web service composition involves combining and coordinating a set of web services with the purpose of achieving functionality that cannot be realized through existing services. The goal of this process is to arrange multiple services into workflows supplying complex and specific user's needs. Hence automating the composition is a really complex and comprehensive problem [1, 7]. In this case the challenge is twofold: (a) How to build a required workflow? and (b) How to discover a more appropriate web service for each node in the built workflow according to user's requirements? In this paper we focus our work to suggest a solution for the second challenge. The first one concerns a future work.

When designing or building a workflow of composite web service statically or dynamically, different approaches are proposed in literature to affect web service to each node of this workflow; for instance each service can be affected in workflow either manually [2, 11], automatically [7, 9] or semi-automatically [3, 10].

For this problem, Said et al. [17] proposes a new SOA architecture called "GenericSOA" that allows dealing with legacy systems problem and enhancing SOA elasticity. The proposed architecture aims to easily integrating the newly developed software components. The main idea behind GenericSOA is to support its users by a set of predefine task templates. These templates can be used in building the new developed services that can be easily integrated in a loosely coupled way to compose the target system.

Natarajan Meghanathan et al. (Eds) : ICCSEA, SPPR, VLSI, WiMoA, SCAI, CNSA, WeST - 2014
pp. 251–263, 2014. © CS & IT-CSCP 2014 DOI : 10.5121/csit.2014.4729

A more appropriate approach is to affect these web services dynamically since published web services is growing more and more, web services are going offline, new services becoming online, and existing services changing their characteristics, user's requirements are contextual and thereafter not static, dealing with failures that may occur when web services are not available, etc. Consequently and due to the increase of published web services, finding the suitable WS that satisfies the user goals among discovered web services still needs deep investigations. Certainly, QoS requirements represent a more appropriate and decisive factor to distinguish similar WSs. A lot of research efforts in this direction have been made but are still limited due to the complexity and diversity of QoS constraints. In this paper we propose a seamless way to compose web services based on abstract process model representing a needed workflow.

Assuming that for each node a collection of web services have been discovered from web service registry depending on user's requirements, so choosing an appropriate one for each node requires in addition to take into consideration their relationship in order to resolve any conflicts and/or inconsistencies between linked web services. Since inconsistencies may occur at runtime, it may be necessary to predict such events to ensure that the composition will run correctly. An important challenge in providing an automatic web services composition facility is dealing with failures that may occur, for instance as a result of context changes or missing service descriptions.

To affect a more adequate web services in composition taking in consideration their correct relationship, we present in this paper the design and implementation of a framework for web services composition. Experimental evaluation demonstrates that the framework provides an efficient and scalable solution. Furthermore, it shows that our framework transforms goals successfully, increasing the utility of achieved objectives without imposing a prohibitive composition time overhead.

In this paper we suggest a web search composition engine that has the faculty to compose automatically web services based on their popularity. We propose in the first hand a workflow design that can be used to draw the composition diagram, and contain a text that the user can use to define his query. The workflow selects the appropriate web services relative to the query based the most popular and generates the composition according the BPEL process model [2] as the result.

The paper is organized as follows; we describe in section II a related work refereeing to automatic web service composition. In section III we describe and explain our proposed algorithm of the automated composition of web services and the rule of how we define the most popular web service by query. In section IV we describe the implementation of our approach in DIVISE and give an evaluation in the system. We conclude in section V.

2. RELATED WORK

Web service composition lets developers create applications on top of service-oriented computing native description, discovery, and communication capabilities. Such applications are rapidly deployable and offer developers reuse possibilities and users seamless access to a variety of complex services. There are many existing approaches to service composition, ranging from abstract methods to those aiming to be industry standards. As defined in the first section, our approach is based on automatic selection and composition, we define two process. In this section, we describe some research proposed for the automatic Web service selection for composition and automatic process for composition.

Recently, Raj et al. [16] propose an approach on identifying the most appropriate service based on the user’s preferences of the requested WS. The given WS description may contain the

parameters, which may have relations with the requested WS of the specific domain in different aspects like name, parameters and types. The domain specific WS classification can be done using Naive Bayes classification algorithm. But this method has some limitations of not considering functionality based classification. The coupling and cohesion properties are not considered in the composed WSs.

2.1. Automate web services selection for composition

The author [4] proposes a novel approach of semantics-based matchmaking, which is named process-context aware matchmaking. This process locates the suitable service during web service composite modeling. During matchmaking, the approach utilizes not only the semantics of technical process but also those of business process of a registered service, thus further improving the precision of matchmaking. The process-context aware matchmaking was integrated with business-process-driven web service composition in a cohesive development environment based on Eclipse. The work describes a way to match web services for composition but doesn't integrate the composition of web services.

To improve the exactitude of a Web service search, Ye and Zhang [9] proposed a method that explicitly specifies the functional semantics of services. They specified a service and a user requirement using object, action and constraints as well as input and output parameters. Utilizing this information, they found a service to satisfy the user requirement. However, they did not consider how the popularity of web services can be applied to service composition.

The authors [11] tried to improve the accuracy of automatic matchmaking of Web services by taking into account the knowledge of past matchmaking experiences for the requested task. In their method, service execution experiences are modeled using case based reasoning. This method can be helpful for improving the exactitude of composite service, but it's still packed of complex problems related to the composition process.

2.2. Automatic Web Services Composition

The University of Georgia implements an extension [5] of GraphPlan [6], an AI planning algorithm, to automatically generate the control flow of a Web process. This extension is does not cover the preconditions and effects of the operations, we also take into consideration in the planning algorithm the structure and semantics of the input and output messages. The approach was presented to solve both the process heterogeneity and data heterogeneity problems. And the system generates outputs, an executable BPEL file which correctly solves non-trivial real-world process specifications. The authors described in parts of their paper the project proposed by BPEL [2] to automate the composition, but neither one of those works propose the automatic selection of web service using the behavior experience named popularity.

In the some context, [8] proposes a composition method that explicitly specifies the uses of functional semantics of web services. Specifically, the proposed method is based on a graph model, which represents the functional semantics of Web services. In this approach, the service functionality of a service is represented by a pair of its action and the object of the action. The information about services is organized and stored in a proposed two-layer graph model. Given a user request, they search for composition paths in the graph model and construct a composite service from the paths discovered. However, the web services selection is not taking in consideration the notoriety to get link in the schema composition.

Liu, Ranganathan and Riabov [10] propose a Web service model in which inputs and outputs of services are expressed using RDF graph pattern, as well as a domain ontology. They improve the

exactitude of composite services without preconditions and effects using semantic propagation based on graph substitution and also they don't take a request user when selecting web services.

3. AUTOMATIC COMPOSITION ALGORITHM

Web services are composed based on QoS metrics [15] by evaluating the utility function and thus maximizing the overall QoS using the hybrid approach in composition patterns. In our approach, we improve the definition presented in [8] that a composite Web service can be defined as a set of transition systems, which has multiple states, arcs and available actions in certain states and represents transitions from an initial state accepting user inputs to a final state providing requested outputs and effects as shown in Figure 1.

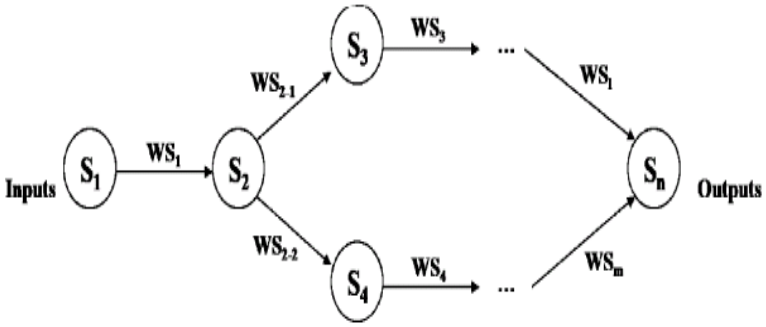

Figure 1: A composite Web Service Representation

In figure 1, "S" represents the web service input variable and "WS" is the web service selected using the query parameter requested by a user while building a graph. As presented in Figure 1, the objective of composing web services is to select required web service for each node in graph based on defined inputs and outputs.

However, the schema mentioned above assumes that every available service is defined by an input and output parameter. For our approach, we present web services as a triple (Input, Query, Output) when Query is the request relative to find the web service. So, we suggest specifying Web Service by its functionality and the sets of I/O as follows:

Web service = (service functionality, input set, output set)

To build the automatic web services composition, we propose, in our algorithm, three fundamental process : (1) Web service composite designer, (2) Web service selection and (3) Web service composite generator. These processes will be presented in details in the following sub-sections. The workflow used is defined in the algorithm defined bellow:

- *Initialize Web Service Composition by defining for each branch of our graph the input, query and output parameter*
- *Search Service Based Popularity: based on query, we calculate the web service criteria value to select the best one*
- *Compose Web Service: define the structure of our composite web service using the web service selected*
- *Generate BPEL File and annexed File: create the web service composite by defining the sequence of activity (Receive, Assign, Invoke and Replay)*

Finally, to complete the composition task, we define the workflow which combines the graph process successively respecting the design produced in the first process. The process selection can be used as a web service discovery to get the service responding to request users sorted by popularity. Note, in our approach the popularity is relative to two criteria, the first one is relative to the frequency of use and the second one will be linked to the appropriate web service.

3.1 Web service composite designer

It's important to begin the web service composition to propose a web service workflow designer. The result proposed in our approach is to generate a BPEL File presenting the web service composite, this module will be used by the client to define the sequence of input, output and a query of Web Service. The architect of our design follows the diagram shown in figure 1; the difference is for each "Invoke" activity in the annotated BPEL in a BPEL Design [13] is that it will be presented by an input text to write the query. In particular, this query information will be used to select the web services in the next process. So, our designer process allows user to draw a sequence of web service call specifying the input and output parameter of each one. it will take a decision on its final result to define the logical orchestration.

3.2 Web Service Selection

This process presents the selection of web services using the user's query. There are two methods for this selection. In the first hand we discover the web service responding the query defined in the input "S", the WS registry will give us a list of result this list will be shorted by the popularity and in the second hand we choose the web service most popular that we present in our workflow to define the "PartnerLink" role defined in the BPEL Process [13].

As defined in the first section, our approach is based on web service selection to automate the composition. In the figure 1, we define the "WS1" designed the first web service , this one will be selected using the frequency presenting the number of web services uses divided by the number of month the formula (1) proposed is [14] :

$$Nb(Invoke(WS))/Nb(Month) = frequency \qquad (1)$$

In the second selection, we use the same criteria of frequency and we add the dependency of the previous web service selected presented by the behaviour experience of the link. There are two methods:

- The first one defined in (2) the number of link between the previous web services selected.
  Nb(Link(WS1, WS2)) = Notoriety
- The second one in (3) but when the previous web service concern two or more the method will be used for the link with the both of them.
  Nb(Link(WS1, WS2, WS3, … WSn) = Notoriety

The process selection provides for each query proposed by user the best web service. In the next step, we extract the web service information that's will be necessary for generating the composite web service as the "accesspoint", the operation and the input and output parameters.

3.3 Web Service composite Generator

The process of web service composition generator is based on creating a BPEL file present the result of composition. This process will be used in two steps. (a) in the first time, we build the BPEL structure based the design provided by the users (b) in the second hand, we execute the

web service selection to add for each invoke tag the web service chooses and we describe the "PartnerLink" tag and "Import" tag to finally generate the web service composite presented by the BPEL File.

4. IMPLEMENTATION AND EVALUATION

The implementation of our approach is based on the BPEL definition [13], because the BPEL process is the most complete and popular language to generate web service composite. In order to test the effectiveness of our proposed algorithm, we have used our implemented framework Discovery and Visual Interactive Web Service Engine (DIVISE) to expose an evaluation of our approach to compose web service by implementing the different algorithms exposed in section III.

Note that DIVISE is an engine that's has the advantage to discover a required simple, composite or semantic web service and to help user to select the more appropriate Web service from a returned list. This list contains in addition to classical web service information a rate of its previous invocations defined as frequency or detailed description detailed of web service, which is useful for calculating the number of link used between web services [14]. This tool is written in Java and mainly built on the Eclipse- frameworks EMF and GEF and thus is also realized as a set of Eclipse.

4.1 Process Implementation

Referring to our algorithm, we propose to create the orchestration of the web service result in four steps. Each one is presented in a principal class executing as an action as presented bellow. In this section, we describe the role of each class and the orchestration of our automatic composition proposition.

- *InitializeWSCompositeAction*
- *SearchServiceBasedPopularity*
- *ComposeWSAction*
- *GenerateBPElFileFromObject*

The designer starts to receive a sequence of request from the user; each request must contain input, query and output parameter. The input and output parameter have to be used successively to define in the last a tree. After each request has been received, the process initializes a vector containing a triple parameter for each web service asked (Input, query, output). Our process control progressively the logic defined in the sequence to get in the first one input as receive parameter and in the last on output as a replay parameter.

Once the model is finished; the schema will be sending to execute the workflow presented as a set of activities. To orchestral this process, we define each level (WS) as a row which contains a set of attributes. This presentation will be used to add web service and proposes in our form the input text relative to each attributes to define the value if necessary. There are some attributes that's can be inserted automatically under the generator and it's not will be presented to user as the url of location or the partner link Etc.

```
public class InitializeWSCompositeAction extends Action {

    public String execute(Map params, HttpServletRequest request, HttpServletResponse response) throws Exception {

    try {
            //Reload the processName and the fileName
            String fileName = (String)request.getParameter("fileName");
            String processName = (String)request.getParameter("processName");

            //Add the file name and process name in request attributes
            if(fileName != null) request.setAttribute("fileName",fileName);
            if(processName != null) request.setAttribute("processName",processName);


            //design vectWSShema contaings hashtable of each WS
            //Vector contains inputs variables {{S1},{S2,S3}}
            String vectWebService = request.getParameter("vectWebService");
            Vector vectWSSchema = new Vector();
            if (vectWebService != null && !vectWebService.equals("null") && !vectWebService.equals("")) {
                vectWSSchema = Utils.explodeList(vectWebService, "|", null, true);
            }
```

Figure 2 : Initialize Web Service Schema Composition

The second process is relative to web service composition is the selection of web service based on popularity. For this process, we implement the algorithm defined in the previous section to calculate the Web Service Popularity Score by query. We create a method called "CalculateWSPSFromQuery" taking query as a parameter and we choose the web service that has the best score to propose it in our composition. To integrate this web service, we also need some necessary parameter defined in the file description that's we extract as the operation, input, ouput parameters etc.

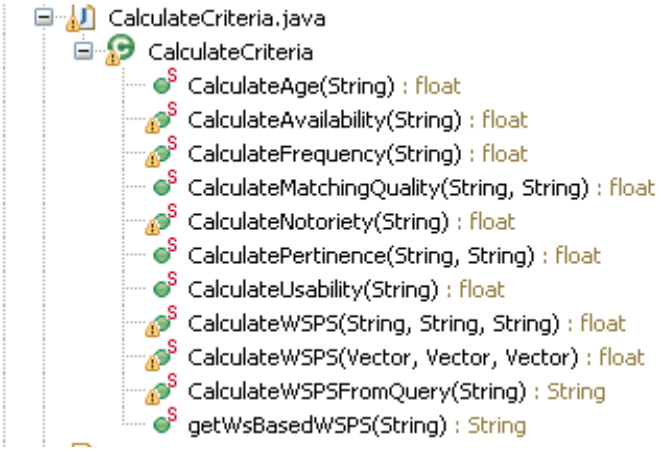

Figure 3 : Methods to calculate Quality Criteria

Based on the schema defined by user and the list of web service selected using the popularity, we create the composition action following the BPEl Model. Howver, we create a BPEL Structure as defined in the figure 4. In this object, we cover the activities as Receive, Invoke, Assign and Replay to design the sequence of the process and the tag relative to import and variable. To resume, our object contains all BPEL tag defined in the XML definition as presented [12].

```
public class BPELStructure {
    // File Name will be used
    //to generate the BPEl File
    public String fileName;
    //Tag <process>
    public Hashtable processName;
    //Tag <import>, this tag will
    //be generator in the composition generator
    public Vector vectImport;
    //Contains Tags <partnerLinks>,
    //this tag will be generator in the generator
    public Vector vectPartnerLink;
    //Contains Tag <variables>,
    //this tag will be generator in the generator
    public Vector vectVariable;
    //Contains Activity as (<Sequence>,<Invoke>,
    //<Receive>,<Replay>,<forEach>,<While> Ect..)
    public Vector vectActivity;
```

Figure 4 : Object of BPEL Structure

This structure defined in figure 4 will be used in the composition process as explained later.

```
public class ComposeWSAction extends Action {

    public String execute(Map params, HttpServletRequest request, HttpServletResponse response) throws Exception {

        String result = "";
        try {

            Vector vectImport = new Vector();
            Vector vectPartnerLink = new Vector();;
            Vector vectVariable = new Vector();;
            Vector vectActivity = new Vector();;

            //construct Object BPEl Structure and generate File
            BPELStructure bpelObject = new BPELStructure();
            //get fileName
            String fileName = (String)request.getParameter("fileName");
            bpelObject.setFileName(fileName);


            //get processName
            String processName = (String)request.getParameter("processName");
            Hashtable rowProcess = new Hashtable();
            rowProcess.put("name", processName);
            rowProcess.put("targetNamespace", "http://"+processName);
            rowProcess.put("suppressJoinFailure", "yes");
            rowProcess.put("xmlns:tns", "http://"+processName);
            rowProcess.put("xmlns:bpel", "http://docs.oasis-open.org/wsbpel/2.0/process/executable");
            bpelObject.setProcessName(rowProcess);
```

Figure 5 : Compose Web Service Action

The class defined in the figure 5 allowed composing web service based on the schema proposed by user. This class, in the first hand, defines the process and their attributes as a map. The result is an instance of “BPELStructure” containing the import WSDL, the list of partner list, the variable used by each web service and finally the sequence of activity. Each activity is presented as a map containing key, values of Activity attributes defined in the BPEL language [12].

```
public static String generateBPElFileFromObject(String fileName, BPELStructure bpelObject) throws Exception {

    try
    {
      DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
      DocumentBuilder docBuilder = docFactory.newDocumentBuilder();

      //root elements
      Document doc = docBuilder.newDocument();

      //process elements
      Element rootElement = doc.createElement("bpel:process");
      Hashtable processRow = bpelObject.getProcessName();
      Enumeration e = processRow.keys();
      while (e.hasMoreElements()) {
          String key = (String) e.nextElement();
          if (processRow.get(key) != null) rootElement.setAttribute(key, (String) processRow.get(key));
      }
      doc.appendChild(rootElement);

      //import elements
      Element importList = doc.createElement("bpel:imports");
      rootElement.appendChild(importList);

      Vector vectImport = bpelObject.getVectImport();
      for (int k =0; k <vectImport.size(); k++){
          Hashtable rowImport = (Hashtable) vectImport.elementAt(k);
          Element importChild = doc.createElement("bpel:import");
          //Tag Attribute defined in hashtable
          e = rowImport.keys();
            while (e.hasMoreElements()) {
                String key = (String) e.nextElement();
                if (rowImport.get(key) != null) importChild.setAttribute(key, (String) rowImport.get(key));
            }
          importList.appendChild(importChild);
      }
```

Figure 6 : Method to generate BPEL File

For the implementation of our approach, we have defined a class defined “ComposeUtils” containing each methods useful for the execution of process.

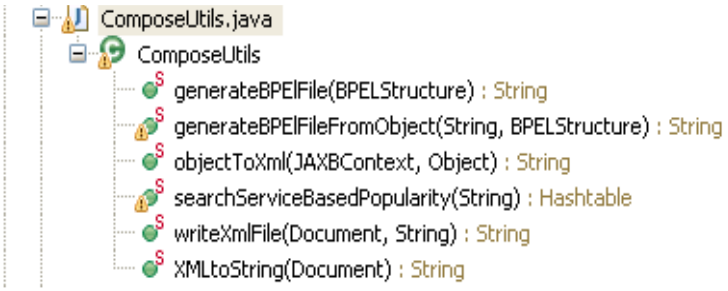

Figure 7 : Utilities for composition process

4.2 Experiment

For improving the efficiencies of our algorithm, we define bellow an example to compose web services using our DIVISE. We have added in the web application menu to compose web services. This link proposes to users to design the workflow process by adding web service request.

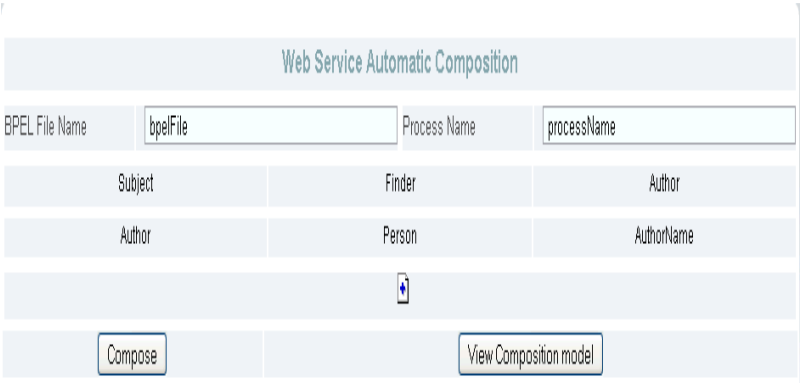

Figure 8 : Web Service Composite Designer

The frame defined in the figure 8 is proposed to design the workflow that's will be used to compose web service. We have choose to use a link (presented as +) to add web service. This link open a window that's allowing user to add a parameter of web service to be added as presented in figure 9. A refresh of the frame is also sending that saves the web service and proposes to add another. We also, in the last submit operation, a control to the workflow to ensure that the sequence designed is correctly follows and each variable are used as input and output.

Figure 9 : Frame to add Web Service in model

The final result is presented as a tree containing the sequence web service and their parameters classed from the first to the last. The figure 10 presents the model composition.

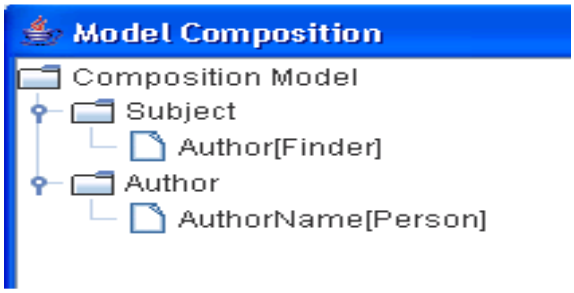

Figure 10 : Tree model composition

After drawing the process, user sends his proposition by clicking compose button. The result of composition is generated in a file containing the orchestration BPEL of our composition process as presented bellow.

Result of Automatic Composition (Bpel File Generated)

File : d:\bpel\bpelFile.bpel

```
<?xml version="1.0" encoding="UTF-8"?>
<bpel:process name="ComposeProcess" suppressJoinFailure="yes"
    targetNamespace="http://ComposeProcess"
    xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable" xmlns:tns="http://ComposeProcess">
    <bpel:imports>
        <bpel:import importType="http://schemas.xmlsoap.org/wsdl/"
            location="" namespace=""/>
        <bpel:import importType="http://schemas.xmlsoap.org/wsdl/"
            location="http://localhost:8080/ServiceProvider/services/PersonService" namespace="http://webservice.ucam"/>
    </bpel:imports>
    <bpel:partnerLinks>
        <bpel:partnerLink myRole="TODO" name="FinderPlnk"
            partnerLinkType="TODO" partnerRole="TODO"/>
        <bpel:partnerLink myRole="TODO" name="PersonPlnk"
            partnerLinkType="TODO" partnerRole="TODO"/>
    </bpel:partnerLinks>
    <bpel:variables>
        <bpel:variable messageType="TODO" name="subject"/>
        <bpel:variable messageType="TODO" name="autor"/>
        <bpel:variable messageType="TODO" name="autor"/>
        <bpel:variable messageType="TODO" name="autorName"/>
    </bpel:variables>
    <bpel:sequence name="main">
        <bpel:receive createInstance="yes" name="receiveInput"
            operation="process" partnerLink="client"
            portType="tns:ComposeProcess" variable="subject"/>
        <bpel:assign name="Assign" validate="no"/>
        <bpel:invoke inputVariable="subject" name="Invoke"
            outputVariable="autor" partnerLink="FinderPlnk" portType="autor"/>
        <bpel:assign name="Assign" validate="no"/>
        <bpel:invoke inputVariable="autor" name="Invoke"
            operation="getPerson" outputVariable="autorName"
            partnerLink="PersonPlnk" portType="autorName"/>
        <bpel:replay name="replyOutput" operation="process"
            partnerLink="client" portType="tns:ComposeProcess" variable="autorName"/>
```

Figure 11 : BPEL File generated

From the result frame, we display the BPEL file generated to user that he can, also, download from the link in the top of frame. In our framework, we can invoke our process from the Web Service Invoke module [15] using the file URL.

5. CONCLUSION AND FUTURE WORKS

The contribution of this paper deals with web service composition, particularly building a deployable composition based on abstract process model (abstract graph). The proposed approach is to assign for each node in the defined abstract graph a required web service meeting user requirements and consistencies between composed web services.

In addition we have developed a prototype system and implemented in our DIVISE framework to improve our approach. This prototype has the advantage to compose web services from a design and select the web services deployed in our log Database filtering by frequency and dependency. Our automatic process proposes to user an interface to create and design a BPEL system that defines the sequence of the activities with query input. Each query will be used to select the best web service based on its popularity. After selection the framework DIVISE generates a BPEL code related to build composite web service.

The future work is to integrate in our approach all activities presented in the BPEL Process and define the pre-conditions or effect, etc.

REFERENCES

[1] P. Bartalos and M. Bielikova, « AUTOMATIC DYNAMIC WEB SERVICE COMPOSITION: A SURVEY AND PROBLEM FORMALIZATION », Computing and Informatics, Vol. 30, 2011, pp. 793–827.

[2] M.Pistore, P. Traverso, P. Bertoli and A. Marconi. "Automated Synthesis of Composite BPEL4WS Web Services", in IEEE Intl Conference on Web Services (ICWS'05). 2005

[3] N. Vukovi c, "Context aware service composition", University of Cambridge, Technical Report UCAM-CL-TR-700, October 2007

[4] W. Han, X. Shi and R. Chen, "Process-context aware matchmaking for web service composition", Journal of Network and Computer Applications, 2008, pp. 559–576

[5] Z. Wu, A. Ranabahu, K. Gomadam, A. P. Sheth and J. A. Miller, « Automatic Composition of Semantic Web Services using Process and Data Mediation »,Technical Report, LSDIS lab, University of Georgia, February 28, 2007

[6] S. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach", Pearson - International edition, 2010

[7] M. Pistore, P. Traverso, P. Bertoli and A. Marconi, « An Approach for the Automated Composition of BPEL Processes », 6789@ ABCDE FGHC6D• I, 2005, p. 24

[8] D. H. Shin, K. H. Lee and T. Suda, "Automated generation of composite web services based on functional semantics", Web Semantics: Science, Services and Agents on theWorldWideWeb, 2009, pp. 332–343

[9] R. Akkiraju, A. Ivan, R. Goodwin, B. Srivastava and T. Syeda-Mahmood, "Semantic matching to achieve web service discovery and composition", Proceedings of CEC/EEE'06, IEEE Computer Society,Washington, DC, 2006, p. 70.

[10] Z. Liu, A. Ranganathan and A. Riabov , "Modeling web services using semantic graph transformations to aid automatic composition", Proceedings of ICWS'07, IEEE Computer Society,Washington, DC, 2007, pp. 78–85.

[11] D. Thakker, T. Osman and D. Al-Dabass, "Knowledge-intensive semanticweb services composition", Proceedings ofUKSIM'08, IEEEComputer Society,Washington, DC, 2008, pp. 673–678.

[12] Eclipse BPEL Project. Eclipse BPEL Designer. http://www.eclipse.org/bpel/

[13] A.Arkin, S. Askary, B. Bloch, F. Curbera, Y. Goland, N. Kartha, C. K. Liu, S. Thatte, P. Yendluri and A. Yiu, "Web Services Business Process Execution Language Version 2.0", Proceedings of the 13th international conference on World Wide Web, Newyork, USA, 2004, pp. 621 – 630.

[14] S. Elfirdoussi, Z. Jarir and M. Quafafou, « Discovery and Visual Interactive WS Engine based on popularity: Architecture and Implementation », International Journal of Software Engineering and Its Applications, 2014, Vol 8, No.2, pp 213-228.

[15] M.Rajeswari, G.Sambasivam, N.Balaji, M.Saleem Basha, T.Vengattaraman, & P. Dhavachelvan, «Appraisal and analysis on various web service composition approaches based on QoS factors»,Journal of King Saud University-Computer and Information Sciences, 2014, vol. 26, no 1, pp 143-152.

[16] T. RAJ, TF Michael, K. RAVICHANDRAN, K. RAJESH, «Domain Specific Web Service Composition by Parameter Classification Using Naïve Bayes Algorithm», World Applied Sciences Journal , 2014, vol 29, pp 99-105.

[17] M. SAID,M. HAZMAN, H. HASSAN, et al. «GenericSOA: a Proposed Framework for Dynamic Service Composition», International Journal of Computer Science Issues (IJCSI), 2014, vol. 11, no 2, pp 94-99.

AUTHORS

Selwa ELFIRDOUSSI

Has obtained a diploma of Engineer in Software Engineering from ENSIAS School of engineering, Mohamed V Souissi University, Rabat, Morocco in 2000. Actually, she's a PhD student at Faculty of sciences, Cadi Ayyad University in Marrakech, Morocco since 2008. Her research interest is focalized on service-oriented computing and Web service technologies.

Zahi JARIR

Zahi JARIR Received his postgraduate degree in computer science in 1997 on Natural Language Processing at Faculty of Sciences in Rabat, Morocco. From 1997 to 2006, he was assistant professor at Faculty of sciences, Cadi Ayyad University in Marrakech, Morocco. In 2006, he received academic accreditation from Cadi Ayyad University. Currently, he is a professor of Computer Science at Faculty of Sciences of Cadi Ayyad University. His research interests at LISI laboratory lie mainly with the field of Service-oriented computing and technologies, Cloud computing and security, Computational reflection and Meta level architectures, Adaptive and Mobile Middleware, and Customization techniques of Web Services and Applications.

Mohamed QUAFAFOU

Mohamed QUAFAFOU did his PhD Thesis in 1992 on Intelligent Tutoring Systems at INSA de Lyon, France. From 1992 to 1994, he was ATER at INSA de Lyon and than at Nantes Faculty of Sciences. From 1995 to 2001, he was assistant professor at the Nantes University. During that period, he developed research on Rough Set Theory, concepts approximation, data mining, web information extraction and participated actively with France Telecom to design a new web system dedicated to French web analysis for discovering emergent web communities. He was also chief-scientist at GEOBS where he headed the Geobs Data Analyzer project, which was developing a spatial data mining systems with application to environment, marketing, social analysis, etc. From September 2002, he was professor at the Avignon University and moved in 2005 to the Aix-Marseille University where he joined the LSIS CNRS and leads research on Data Mining Theory and Applications focusing on new contexts for learning (crowdsourcing, interconnected data, and big data) with application to user's behavior analysis, web services, cloud automatic auto-scaling, social network analysis, etc.