A Flexible Approach to Compose Web Services in Dynamic Environment

Kaouthar Boumhamdi, Zahi Jarir
Computer Department, Faculty of Sciences, Cadi Ayyad University

Abstract

Web services have received much interest due to their potential to design and build complex inter-enterprise business applications. A particular interest concerns dynamic Web services composition that offers the opportunity for creating new Web services at runtime from those already published in UDDI registry. However, this composition requires necessarily quantifying criteria for Web service selection to build a more appropriated one. The objective of this paper is to suggest a solution for this challenge by proposing a flexible architecture for dynamic Web services composition related to user's requirements and preferences in harmony with the availability of more critical resources. In addition, this architecture has the ability to re-configure at runtime the created composite service, if some failures happen, in order to maintain a better quality of composite Web service.

1. Introduction

Services Oriented Architecture (SOA) [1] uses the concept of service as an elementary brick to assemble complex systems. It provides means for the self description, announcement, discovery, interaction and usage of services. These services are loosely-coupled by definition, and complex services may be built by using composition mechanisms.
The Web services composition (WSC for short) can be done in a static or dynamic way. The static composition allows the requestor to create an abstract model that should be respected during the execution of this Web service. While the dynamic composition enables selecting the atomic Web services automatically and combines them to create an unlimited number of new Web services.

The composition is frequently specified using the Web Services Business Process Execution Language (BPEL for short) [2], it is an XML based language that provides the user with variables, conditionals, loops, process correlation, asynchronous messages and exception handling. However, in practice, WSC task is often affected by changes happened continuously in the WSC execution environment and also guided by user's requirements and preferences. In our case, WSC execution environment includes both user execution context, middleware context and web service execution context. Therefore, WSC approaches must take into account these changes to adapt automatically the composed Web service. In this direction, few studies have addressed this issue. To this end, we propose in this paper an architecture based on a flexible approach to meet the more critical situation faced by WSC:
(a) Failures that can occur in a composed business process execution and particularly when a service becomes unavailable, which can cause a downtime in the overall process execution. This can be due to a number of reasons like Network failure; the service has been withdrawn or changed to some form incompatible to the previous one.
(b) Incapability of BPEL engine to monitor running process and thereafter to decide which services used is candidate to a replacement or suppression from the used composition when a failure is happened. In this direction, the issue is thereafter how to ensure the availability of these services and also how to ensure a good execution of the business process at runtime, without editing and redeploying the process which implies a downtime of the overall system. Typically, a dynamic replacement of Web services in the process, which is fastidious task, consists on using a more appropriate service among services implementing the same functionality as the failed one.

The approach we present in this paper focuses mainly to monitor a running BPEL process in order to detect failed web services implicated by a process. When failures are detected, it will thereafter resolve dynamically those failures in BPEL scenario executions by invoking replacement of the failed services by the more appropriate ones which are functionally similar according to the current execution environment.

The rest of the paper is organized as follows. Section 2 presents briefly some interesting techniques of dynamic composition of Web services. Section 3 describes our approach for adapting on the fly composites Web services according to user's preferences and to the availability of critical resources and its implementation. Finally, Section 4 presents a conclusion and some perspectives.

2. Related work

The dynamic composition in SOA is defined as a process to select and to combine a set of atomic or composite Web services automatically. In this paper, our interest focuses on a dynamic composition of services in volatile environment [3], by taking into account user's preferences and requirements, and the WSC execution environment, like availability, reliability, cost, execution time, robustness, limited bandwidth, scarce resource, accessibility, performance, etc.

However, Web services composition is faced with two main challenges: (i) How to create the most appropriate composite Web service according to the user requirements and his context of use from a range of Web services published in UDDI? (ii) How to reconfigure this composition, if a failure occurs during execution, by ensuring a better quality of service, according to the user's preferences and/or the availability of some critical resources?

This challenge of composing Web services dynamically was handled by several researches. To more evaluate these researches, we introduce five criteria to better evaluate WSC approaches which are: Execution monitoring, QoS Modeling, Transaction Support, Recursive composition and User Interaction. These criteria are more detailed in the following sub-section.

2.1. Web service composition requirements

To build a new composite Web service and to adapt it at runtime, the used composition process must take into account some important criteria. These criteria are summarized as follow:

- Execution monitoring [4] that aims to focus on monitoring at runtime the state of composite service, to detect and to identify changes related to the Quality of Service (QoS) of each among those considered by the composition process. In this case, we identify three kinds of monitoring: Centralized, distributed and hybrid execution. The Centralized execution imposes that the server is the central scheduler that controls the execution of the composite Web service components. The distributed execution requires the participating Web services to share their execution context and also their hosts, to coordinate with each other. While the hybrid form has the advantage to control a set of services both in distributed execution form and centralized execution form.

- QoS Modeling, which presents the structure that, provides the description of different QoS aspects such as availability, reliability or performance [5].

- Transaction Support, [6] which helps to improve the reliability and fault-tolerance of the Web services currently used to guarantee their interactions. It supports two kinds of transaction: (1) short-duration transaction, called atomic transaction and (2) long transaction. In general, it's not always possible to ensure transaction atomicity, consistency, integrity and durability for a long transaction of a Web service namely ACID.

- Recursive composition [7], which defines a mechanism to detect resolves and recover faults of composition process on-the-fly in the dynamic environment.

- User Interaction [8] that allows user to supervise the composition process, starting from the generation of an abstract plan, to the creation and the deployment of an executable workflow in a runtime system.

2.2. Related dynamic Web services composition approaches

In this Section, we introduce some interesting dynamic Web service composition techniques.

The work in [9] introduces the approach of Model Driven Service Composition, which is based on dynamic service composition. It facilitates the management and the development of dynamic service compositions. The authors define four components for composition process: (a) the definer that defines the composition rule related to user's request, (b) the scheduler that develops a set of concrete compositions workflow, (c) the user that selects an alternative, and (d) the executor that monitors the service execution.

However, this approach supports neither automatic recovery from failure states nor monitoring mechanisms to validate the generated composition.

In declarative composition presented in [10], composite services are generated from a high-level declarative description. The technique uses composability rules to determine whether two services are composables. This approach consists of two phases. The first phase takes an initial situation as starting point and the desired goal, and then constructs a plan to achieve the goal. This phase is realized using PDDL (Planning Domain Definition Language). The second phase chooses one and the best generic plan, discovers appropriate services, and builds a workflow out of them. This phase may be realized by using existing process modeling languages, such as BPEL.

However this approach supports no interaction with users at runtime, which is necessary if user's preferences changed.

Authors in [11] are proposed an approach based on runtime reconfiguration using wrappers. The rule of these wrappers is to make the component capable

to interact with a new component, when it contains an interface not compatible with existing component at runtime. Nevertheless, several problems in wrappers exist: (1) Type conflict between existing and newly introduced components should be resolved by using common parent types between component and wrapper, (2) Current component models do not meet the requirements to support unanticipated runtime reconfiguration. To meet these challenges, this approach introduces a reconfiguration manager. This manager controls the wrapper, by extracting the component type information from newly introduced component, before injecting the wrapper. This information enables the wrapper to adapt type component. Also the reconfiguration manager controls how a wrapper is composed with existing component. It decides when and the order at which the wrapper should be executed.

However, the composition is made in a complicated recursive manner, and sometimes, type conflict between passed parameters between services may occur and also it does not support transaction.

A Runtime service adaptation [12] involves adapting components into new components or services by changing the interface or behavior of the component at runtime, which allows making incompatible components composable. This approach is subdivided on two types of runtime adaptation: (1) Superposition, which is a technique that enables software engineer to impose predefined, but reconfigurable, types of functionality on the operations that a component can support, (2) Type-Safe introduces typed delegation as a means for components to interact in addition to simple message passing. However, this approach does not allow execution monitoring, transaction, recursive composition and interaction with users at runtime.

A composition language [13] provides a means to define higher-level abstractions that better describe component composition. A composition language is a combination of (a) an Architectural Description Language (ADL), which is used to specify the component architecture, (b) a scripting language, which is using to define various configurations of components based on architectural style, (c) a glue language, which is used to specify a component adaptation strategy and (d) a coordination language to specify and to configure the coordination mechanisms and policies for concurrent and distributed components. Although, this approach does not support: transaction, recursive composition and interaction with users at runtime.

A Workflow Driven Composition [14] defines an abstract process based on business and process constraints, includes the dependency of atomic services, the user's preference and the execution context, and finally allows to generate an executable process, using language BPEL4WS, that answers the user query.

However, the authors did not address any issue related to recursive composition and user interaction properties.

Adaptation technique coming from the workflow area, based on execution monitoring [15], allows monitoring of BPEL process according to QoS and replacement of existing partner service, by intercepting SOAP messages to enable services to be exchanged during runtime. They presented the VieDAME framework that uses a selector component to choose the most adequate service: in VieDAME each service in a BPEL process can be marked replaceable and indicate an alternative service that can be configured and invoked instead of the original service.

Thereby, each service and all of its alternative service's endpoints are stored in the VieDAME service repository, and to use alternative services instead of the original partner service they use Transformers component that compensates the interface mismatch between the original service and the alternative. However, this approach doesn't support interaction with users at runtime which are generally an important entity in composition process.

Approach for dynamically resolving exceptions occurring in BPEL scenario executions [16]. This approach caters of the exceptions resolution generated due to system faults, such as network fault. The proposed approach uses the Alternative Service Operation Binding (ASOB). Framework, which is a middleware, based approach for dynamically intercepting any failures and resolves them by invoking operational replacement services that are equivalent to the failed one; the reply is returned to the WS-BPEL scenario.

Nevertheless, this approach supports no interaction with users at runtime, when the new criteria for replacement service selection.

To more compare the eight propositions evoked in this section, Table 1 summarizes their characteristic.

In accordance with the Table 1 which compares the different evoked approaches for Dynamic Web service composition, we note that no approach support all or the majority of criteria, such as execution monitoring, QoS modeling, recursive composition and support user interaction. This motivates as to interest to dynamic Web service composition, and to suggest a solution to this challenge by taking into account these criteria in the composition process.

Table 1. Summary of evoked approaches

Composition Techniques	Execution monitor	QoS Modelling	Transaction support	Graph support	Recursive composition.	Language/ FrameWork	Support user interaction in the composition process
Model driven web service composition	Non	Yes	Yes	Yes	Non	BPEL4WS. The OCL (Object Constraint Language)	Non
Declarative service composition	Yes	Yes	Non	Yes	Yes	PDDL, XSRL BPEL, FUSION	Non
Runtime reconfiguration using wrappers	Yes	Yes	Non	Yes	Non	Proteus	Yes
Runtime service adaptation	Non	Yes	Non	Yes	Non		Non
Composition language	Yes	Yes	Non	Yes	Non	ADL, scripting language, a glue language, et a Coordi-nation language	Non
Workflow-driven compositions	Yes	Yes	Yes	Yes	Non	BPEL4WS	Non
Adaptation based on execution monitoring	Yes	Yes	Yes	Yes	Yes	VieDAME	Non
Approach for resolving exceptions occurring in WS-BPEL	Yes	Yes	Yes	Yes	Yes	ASOB	Non

3. Proposal for a dynamic composition approach

In this Section, we suggest our solution to meet the lack presented by the evoked approaches. This solution concerns dynamic Web service composition by taking into account the user's requirements and preferences according to the availability of some critical resources related to the WSC execution environment. More precisely, this approach integrates a mechanism to detect at runtime, if a failure occurs during the execution of the composite service, to resolve and to recover the already generated execution plan in an appropriate way to ensure a better QoS.

This section is organized as follow. We start with a description of our architecture. After that we present in more detail the main components of this architecture which are surrounded by the red box in Figure 1. The last sub section is dedicated to presenting a case study to more explain the behavior of the proposed architecture.

3.1. Description of our architecture

The main objective of our architecture (cf. Figure 1) is to make the composition of Web services more flexible and adaptable according to the user profile.

The architecture that we propose is modular and extensible with the advantage to ensure the most appropriate composite Web service according to the availability of resources to preserve a better QoS. In addition, it supports also recursive composition and user interaction properties.

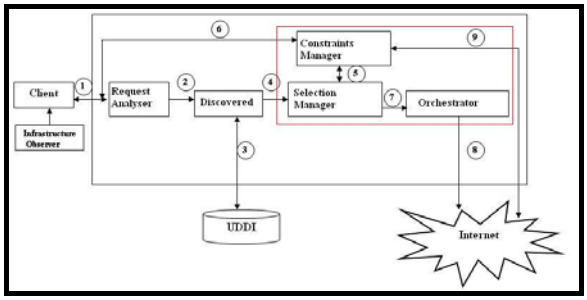

Figure1. Proposed architecture

This architecture is modularized into five distinct components which are presented as follow:

A Query Analyser aims to analyze the client's request to identify his requirements. Then it proposes the required functionalities in term of Web services. This component is based on a phase of lexical and semantic analysis to allow users to use a formalism of the highest level. This work is under work by another subject in our team. A Discovery Component allows discovering and locating the basic Web services published in the UDDI directory related to the required functionalities detected by Query Analyser. This component then communicates the list of Web services to a selection manager component.

A Constraints Manager Component is responsible to monitor the availability of the critical resources at runtime, and to detect if a failure occurs during the execution of the composite service. The purpose of the Selection Manager is to generate or regenerate the execution plan, from a communicated list of available services discovered in UDDI registry and in harmony with information received from the Constraints Manager. The Orchestrator has the ability to handle the execution plan in BPEL language of the generated Web service composite and also allows monitoring the business process in running environment.

3.2. Constraint manager component

This component based of a monitoring framework, consisting of a collection of probes (CPU usage, battery life, bandwidth measure…), is inspired from a previous developed work [8]. This component is also responsible to detect dynamically changes in user preferences and WSC execution environment. If some changes are detected, it will notify Selection Manager Component to handle the required change.

Recall that WSC execution environment includes all significant changes that can be happened at user execution context, middleware context and also Web service context. User execution context refers for example to user configuration (e.g. configuration of a

used terminal, availability of critical resources, etc.), whereas web service context refers to parameters that concern for instance its availability, QoS, and so on., and middleware context refers to parameters concerns for example network-related characteristics, etc.

3.3. Selection manager component

Selection manager is an implementation of our algorithm presented by sub section 3.5 that determines which are the best services reply to the user's preferences and according to WSC execution environment. Moreover, when the Constraint manager detect a changes in user profile the Selection manager offers fault compensating selectors that retry failed service and return to the Discover component to choose an alternative service with the same functionality as the original service. After, the Selection manager generates or regenerates the execution plan BPEL.

3.4. Orchestrator component

BPEL does not specify any details about process monitoring; it provides a combination of "throw" and "catch" activities for catching unavailability fault and invoking replacement services.

Several problems can occur, among them: (a) if a partner service (partnerlink) becomes unavailable or faulty, the process could call an altering service (typically 2 or 3 alternates will be specified). In the worst case, the altering service is down will result a downtime of the overall system. (b) Time of responsiveness, which is duration to respond to a customer request. It is a matter on how fast or slow each of the involved partner services responds.

The Orchestrator, execute the BPEL plan already generated by the Selection manager that all invoked partner services are available, which enable to optimize the execution time of the business process.

Consequently, The Orchestrator execute the BPEL plan which is the business process coded in BPEL language generated by the Selection manager component, it allows also the monitoring of various QoS attributes (availability) of running BPEL.

3.5. Algorithm for dynamically composing Web services

To handle a dynamic composition process, we have implemented the algorithm presented as follow. Three components are involved in this algorithm, which are Selection Manager, Constraints Manager and Orchestrator (the red frame in Figure 1).

```
SelectionManager

Begin

//The sequence of the services to invoke deducted from the client request
GraphSequencesServices graphSequencesServices;
// Services to discover from UDDI
ListServices listServices ;
//User's preferences from SelectionConstrainte
ListUserPreferences listUserPreferences ;
//The characteristics of the execution context from SelectionConstrainte
ListConstraintsCtextUser listConstraintsCtextUser ;
newcontxtuserconstraintes = False // if change in user's context
newuserPreferencesconstraintes = False // if change in user's Preferences

Iterator iterator = listServices.iterator();

While (iterator.hasNext()) and (newcontxtuserconstraintes = False) and (newuserPreferencesconstraintes = False){
    Service service = iterator.next();
//Compare the values of service properties with the
//constraints and the user context after classify them
//according to the constraints define by the user's profile.

Map map = ValidateClassifyServices (getParametreWSDL(service),
listUserPreferences(service),
listConstraintsContextUser(service)) ;

} // end while

//Choosing the best services verifying the constraints

List listBetterService = getListBetterService (map);

//Generate execution plan according to the graph of
//sequence that can be executed by the Orchestrator

PathBPEL = genaratePathBPEL (listBetterService, graphSequencesServices);

End
```

To be in harmony with changes detected by Constraints Manager Component, when some changes occur in the user's context or user's preferences, we update the list of preferences constraints and/or list of context constraints.

We have used the following pseudo code:

```
If (newcontxtuserconstraintes = true) then
  Update ListConstraintsCtextUser
  Call SelectionManager
Elseif (newuserPreferencesconstraintes = true) then
     Update listUserPreferences
     Call SelectionManager
End if
```

3.6. Travel Reservation Study Case Scenario

To illustrate our approach as supported by our architecture, we present a scenario based on a travel reservation study case.

Suppose that a client requests for a plane ticket, which is conditioned by hotel and excursion reservation. The reservation, imposed by a client, introduces some constraints that concern price, hotel category, choice of airlines, etc.

The user send his request to the Request analyser component, the requester has to be informed well his needs that allows the system identifying the sequence of the services to be invoked. The next step is the Request analyser, which analyses the user's query to determine the required Web services, which are in our context three different Web services, a plane ticket booking, a hotel reservation service and excursion reservation service. It also identifies the sequence of these services to generate the desired composition. In addition, it deduces the requirements related to each Web service. These constraints relate for example the name of the Airline Company, departure and arrival dates, destination of the flight, the price, type and category of hotel, etc.

The discovery component receives a list of basic Web services to be discovered from the UDDI registry. Once discovery processes is completed, the list of Web services, implementing a required functionality, is sent to the selection manager component.

The selection manager component will thereafter select the appropriate Web service from the list taking into account the user profile, through an implementation of our selection algorithm that determines a list of the best available services matches one or more user's requirements. Moreover, during the execution, if the constraints manager component detects a change in the execution context and/or user preferences, it will notify the selection manager component to retry failed service invocations that not answer the requirements, either with the original service or with an appropriate replacement. After it readjust the composition plan by generating a required BPEL plan to be executed by the Orchestrator component.

The final step is the orchestrator component that executes the BPEL plan generated.

3.7. Implementation

Before choosing the adequate environment for developing our architecture, we have compared the existing modeling tools and servers designing, deploying, and monitoring BPEL business process (cf. Table 2 and Table 3).

Therefore, we have chosen the open source tool Eclipse BPEL Designer to model, to design, and to build composite services, the Orchestration Director Engine (ODE) server for deploying and monitoring business processes, also WSO2 Business Process Server (WSO2 BPS) that provides a complete Web-based graphical console to deploy, manage and view processes in addition to managing and viewing process instances.

Table 2. Summary of process modelling tools of BPEL

Graphic tools	Designer	Model	Validation	Runtime Framework	Debug	Generation automatic WSDL	Open Source
Activos	Yes	Yes	Yes	Yes	Yes	Yes	Non
Eclipse BPEL Designer	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Oracle BPEL Designer	Yes	Yes	Yes	Yes	Yes	Yes	Non
Zenflow	Yes	Yes	Yes	Non	Yes	Non	Yes

Table 3. Summary of BPEL Servers

Servers BPEL	Monitoring	Administration	Failure Resolution	Expose the process as SW	Open Source
Active BPEL	Yes	Yes	Yes	Non	Yes
Nova Orchestra	Yes	Yes	Yes	Non	Yes
Apache ODE	Yes	Yes	Yes	Yes	Yes
Oracle BPEL Process Manager	Yes	Yes	Yes	Non	Non
Twister	Yes	Yes	Yes	Non	Yes

Currently, we are under finishing work on developing the two components: The selection manager component and the constraints manager component.

4. Conclusion

The main contribution of this work is to propose a solution for dynamic composition of Web services. We have started by presenting a brief state of the art that includes some important approach to compose Web services in a dynamic way. Thereafter, and based on comparative evaluation of these approach, we have suggested our flexible and modular architecture. The proposed approach has the ability to adapt and to recover on the fly the composed Web services according to the user's requirements and/or the availability of main critical resources. In addition this dynamic reconfiguration is guided to preserve a better QoS by proposing the more appropriate Web service composite.

Currently, we are finishing the implementation of our approach in order to evaluate it with a more complex case study.

5. References

[1] Mackenzie, M.Laskey, K. McCabe, F.Brown, P.Metz, Reference Model for Service Oriented Architecture 1.0. Technical Report wd-soa-rm-cd1, OASIS (February 2006)

[2] OASIS.Web Service Business Process Execution Language 2.0, 2006. URL : http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.

[3] J.Harney and P.Doshi, "Speeding Up Web Service Composition with Volatile External Information",ACM International Conference Proceeding Series, (WWW 2008) Beijing, China,2008, Vol. 292.

[4] Sun, H., Wang, X., Zhou, B. and Zou, P. Research and Implementation of Dynamic Web Services Composition, APPT 2003, LNCS 2834, Springer-Verlag Berlin Heidelberg, pp.457–466.Terai, K., Izumi.

[5] A.L.Blevec, C.Ghedira, D.Benslimane and Z.Jarir, "Exposing Web Services to Business Partners: Security and Quality of Service Issue", International Conference on Digital Information Management(ICDIM), Bangalore, India, 2006, pp: 69-74.

[6] Papazoglou, M.P. " Web Services and Business Transactions, World Wide Web: Internet and Web Information Systems", Kluwer Academic Publishers, 2003, Vol. 6, pp.49–91.

[7] Khalaf, R. and Leymann, F. (2003) Benatallah, B. and Shan, M-C. (Eds.): On Web Services Aggregation, TES, LNCS 2819, Springer-Verlag Berlin Heidelberg, pp.1–13.

[8] Z.Jarir and M.Erradi," Dynamic Personalization in Component-based Application ", Asian Journal of Information Technology, Vol 3, Numero 9,© Grace Publications Network, 2004, pp. 796-800.

[9] S.D.Dustdar, and W.S. Schreiner," A survey on web services composition", Int. J. Web and Grid Services, 2005, Vol. 1, No. 1.

[10] Orriens, B., Yang, J. et Papazoglou, M.P. Orlowska, M.E. et al. (Eds.), "Model Driven Service Composition", ICSOC, LNCS 2910, Springer-Verlag Berlin Heidelberg, 2003), pp.75–90.

[11] A.A. Alamri, M.E. Eid, and A.E. El Saddik, "Classification of the state-of-the-art dynamic web Services composition techniques", 2006, Int. J. Web and Grid Services, Vol. 2, No. 2.

[12] Bosch, J. 'Superimposition: a component adaptation technique', Journal of Information and Software Technology, April 1999, No. 41, pp.257–273.

[13] Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D. L., Mcllraith, S.A., Narayanan, S., et al. "DAML-S: semantic markup for web services", Proceedings of Proceedings International Semantic Web Conference (ISWC), Sardinia, Italy, June 2002.

[14] Giacomo, P. and Williams, S.L. "Workflow: a language for composing Web services", Proceeding of Conference on Business Process Management (CBM 2003), Berlin, Heidelberg, Germany: Springer-Verlag, October.

[15] O.Moser, F.Rosenberg and S.Dustdar," Non-Intrusive Monitoring and Service Adaptation for WS-BPEL", International World Wide Web Conference Committee (IW3C2). WWW 2008, April 21–25, 2008, Beijing, China. ACM 978-1-60558-085-2/08/04.

[16] K.Christos, C.Vassilakis, E.Rouvas and P.Georgiadis, "Exception Resolution for BPEL Processes: a Middleware based Framework and Performance Evaluation", iiWAS2008, November 24–28, 2008, Linz, Austria. 2008 ACM 978-1-60558-349-5/08/0011.

