RC24216 (C0703-005) March 12, 2007
Computer Science

IBM Research Report

An Intelligent Service Composer for Business-level Service Composition

Liu Ying, Wang Li
IBM Research Division
China Research Laboratory
Building 19, Zhouguancun Software Park
8 Dongbeiwang West Road, Haidian District
Beijing, P.R.C. 100094

An Intelligent Service Composer for Business-level Service Composition

Liu Ying, Wang Li
IBM China Research Lab
{aliceliu,wanglcrl}@cn.ibm.com

Abstract

Service Component Architecture (SCA) provides a first class model for building systems using Service Oriented Architecture (SOA). But SCA only focus on functional level service components. Considering a business system includes not only functional level components but also user interface and data level components, this paper presents a unified service composition framework to support business level service composition. This framework covers 3-layer composition, including functional, data, and user interface level service composition. An intelligent service composer based on this unified service composition framework is developed to enable business level service composition by business people under some advanced technologies, including intelligent service components searching, automatic service compliance checking, and template-based service adaptation. An example is presented to illustrate how a business people create a business application by using the intelligent service composer.

1. Introduction

The growing trend in enterprises is to assemble business systems from a set of appropriate web services and no longer be written from scratch. Service Oriented Architecture (SOA) [7] has gained a lot of attention, so that more and more enterprises have started to encapsulate their software components as service components and created their business applications by composing service components. In order to help enterprises easy to create and integrate business applications, Service Component Architecture (SCA) [8] further provides a first class model for building systems using SOA, which can be regarded as a complementary of web services by providing a means of assembling services into a business system, as well as providing a service construction model.

However, current SCA focuses on the connections of functional units of an application. A complete business system obviously should have the user interfaces and data level composition consideration. It is a natural way to take the UI and data level composition into account while assembling a SOA application. In this paper, we present a unified service composition framework which covers 3-layer composition, including functional unit composition, UI flexible binding, and data integration. While creating a business system, the composition of related set of functional components are taken as the starting point, then corresponding UI components can be flexibly bound with the functional components. If the data structure or sources are inconsistent among these functional components, they can be tackled at data level of this framework.

The potential for businesses to be able to interact with each other on the fly is very appealing. In order to quickly and correctly create business application by composing service components, business domain knowledge, such as business logic and service compliance requirements, are demanded. To date, the activity of creating SOA business applications can not be performed by business people because of technique gap. The gap comes from the following four reasons: firstly, the limitation of current service component standard is that it only addresses how to define interfaces but not business semantics, then business people can not understand them; secondly, it is difficult for business people to manually write adaptation while two service components are mismatched; thirdly, facing a large volume of service components, business people could not easy to find the required service components; fourthly, lacking of mechanisms to let business people be aware of non-compliance errors is another challenge.

In order to bridge the gap between existing service composition technology and required business level service composition, we developed an intelligent service composer based on the above mentioned service composition framework to enable business level service composition. This service composer has four important features: intelligent service components searching to enable business people quickly locate the required service components; automatic compliance checking to support service functional and non-functional level service compliance checking; template-based adaptation to enable quickly and easily adapt mismatched service components; and automatic wiring feature to enable quickly connection of multiple service components. This composer is web-based tool, which provides the capability to quickly deploy the composed business service, so that the quick demo can demonstrate the function of composed business service immediately.

The rest of this paper is organized as the following structure. Section 2 presents the overview of the unified service composition framework. In section 3, three important features of the intelligent service composer are introduced, including intelligent component searching, automatic compliance checking, and template-based service adaptation. UI level composition is presented in section 4. In section 5, we briefly introduce what are the critical challenges for data-level composition. In section 6, an example is taken to illustrate how to compose a business application by using the intelligent service composer. Related works of service composition are listed in section 7. We give the conclusion in the final section.

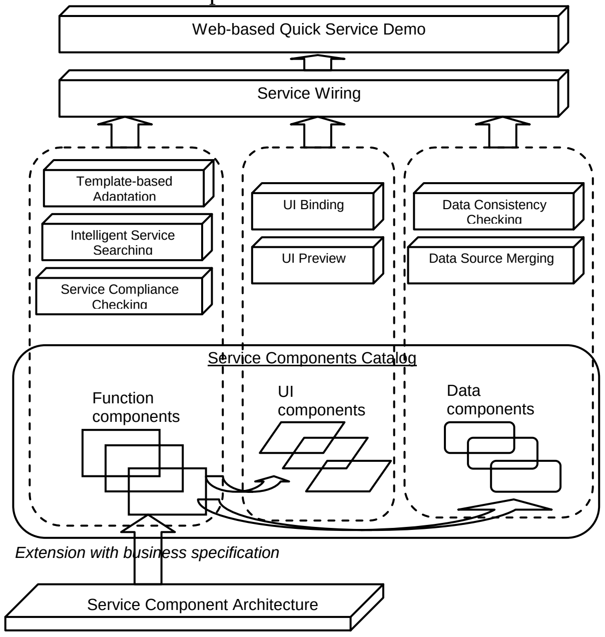
2. Unified service composition framework

A business application is usually composed of three parts, data structure, business logic, and UI. Correspondingly, we define a unified service composition framework from these three dimensions. The overview of this framework is showed in figure 1.

Function components in this framework extend SCA with business specification. The business specification is classified as four kinds: the first one is some meta-information, including name, description, domain category, domain related keywords, associated function components, provider information, and implementation language; the second is some SLA (Service Level Agreement) related properties, including availability, security policy, duration of service, performance metrics (e.g. maximum response time, maximum capacity, throughput), management policy (e.g. government policy, help desker information); the third is business logic constraints; the fourth one is the service delivery mode, such as lease mode (the users can only use the service by web access channel, and the service is charged by per year, per month or per request) and sell mode (the service component is sold to the users). From representation perspective, a service function component is composed of a business specification file in xml format and some jar files which are implementations of the SCA components.

A UI component encapsulates some UI pages, some picture files, and some java files which implement the UI actions. A UI component can be existed independently, but the general situation is that a UI component is glued with a service function component.

A data component includes two parts: one is the data source specification (e.g. database version, xml structure); the other is the data structure definition in xml format. Similar to the UI components, a data component can be defined independently or glued with some function components.



Web-based Quick Service Demo
Service Wiring
Template-based Adaptation
Intelligent Service Searching
Service Compliance Checking
UI Binding
UI Preview
Data Consistency Checking
Data Source Merging
Service Components Catalog
Function components
UI components
Data components
Extension with business specification
Service Component Architecture

Figure 1: Service Composition Framework

Since the target user of this framework is business people, it is necessary to provide some easy-to-use functions to support business people to compose business service. Corresponding to the four reasons to lead the technical gaps, we provide the following key functions to bridge the gap.

How to quickly locate the required service components is the first step. In this framework, some intelligent searching mechanisms are designed based on the features of function components. As the above introduced, the function components are specified with some business characteristics, such as domain related keywords, associated function components, and SLA related properties. Then the business people can search the required service components by keywords and SLA properties. While creating a business application, the associated service components can be located once one component of the business application is found because the association relationships have been specified in the business specification of service components. For example, while composing an *account receivable* application, a function component ‘*create payment account*’ is firstly searched by keywords, the associated components ‘*customer relationship for RA*’ and ‘*payment management*’ can be quickly located because these two components are associated components of ‘*create payment account*’ component.

The compliance judgment of service components no matter from functional of SLA perspectives is a big challenge for business people since they are not technical experts. Automatic compliance checking function is provided in the service composition framework. The details will be introduced in section 3.1.

If the interfaces of two components do not match, to manually write the adapter of these two components are very difficult for business people. We propose to apply template-based automatic adaptation technique to replace manual adaptation.

There are two important features, UI binding and UI preview, at UI composition layer. Through flexible UI binding, users can choose their favorite UI style just-in-time. UI preview function enables users to visually select UI components they want.

As we know, data composition is a quite difficult topic. Fortunately, a lot of existing works related to information integration can be leveraged. In section 5, we will analyze some specific challenges in the service composition but without concrete technologies.

We developed a web-based service composer following the service composition framework. Besides the above introduced function features, the composer provides quick demo capability. When the service composition is finished, the generated runnable files can be automatically deployed in the web platform. The users can directly preview the execution result of the composed service. It is beneficial for business people to immediately make decision whether the composed service satisfies the business requirements.

3. Intelligent features of composer

Three important features of the composer are introduced in this section, including intelligent searching, compliance checking, and template-based adaptation.

3.1. Service intelligent searching

Business people usually know what they want from business perspective, but they have no idea how to find them because of lacking of technical background to understand service interface definitions or complex service specification. However how to quickly locate suitable components and correctly compose them in line with a given business model or to fulfill a specific business requirement becomes a big challenge based on existing web service composition technique.

The intelligent composer provides the *intelligent searching* capability to help business people quickly find out required service components. Actually, there is no specific research difficulties while implementing this capability but leveraging some searching technologies. We particular indicates some aspects that we have considered while designing this capability. Firstly, business people can easily follow some business questions, so that the composer can guide the users to locate the target components step by step under the guidance of those business questions. And these business questions have been internally connected with related service components. Secondly, the organization structure of service components are critical important. The composer is based on a service catalog which aligns with asset repository standard. Then the business people can search component by domain, business rule, provided information, prices, etc. Thirdly, the business specifications of service components play important role in supporting intelligent searching. Users can search service components by following the attributes defined in the business specification, such as environment (operation system, language, system), interface (methods, input/output), SLA (performance, limitations, database usage), and component type, size, domain, etc. Fourthly, user’s feedback is very important for dynamically updating component usage information based on searching historical record, such as service level information, weights of searching algorithm, categorization, etc.

3.2. Service compliance checking

A service component is often developed for a specific business or a kind of businesses. The component provider is quite clear what situation is the appropriate business to adopt this component. Therefore, we propose the provider specify these constraints while delivering this component, including business constraints and SLA properties. Business constraints are specified as some compliance rules, which should be complied in the functional level composition. The compliance rules can be categorized as the following three kinds:

Meta-information matching: The meta-information of a function component has the corresponding constraints while composing it with other components.

For example, the provider of a service component is company A, and one of a compliance rule required that the composed components with it should have the same provider company A. The meta-information matching rule is specified as the following format:

composite. Provider = SAME or composite. Provider = 'Company A'

All the meta-information of a service component can be defined as the properties of the component, so the meta-information matching rules are defined as the assignment on the values of its composite components. The implementation of checking meta-information matching rules is intuitive, i.e. to analyze the business specification files of the composed service component to check whether these rules are complied or not.

Business logic constraints: A function components often includes multiple interfaces. In order to help users easily and correctly assemble application through the wiring of service interfaces, each service components define some business logic constraints to describe the interfaces connection rules. For example, a component '*create receivable account*' has two interfaces: one is *GetAccountInformation(Customer)*, and the other is *GenerateReceivableAccount(Account)*. The business function of this component is to generate receivable account after analyzing the information of customer. The interface *GetAccountInformation(Customer)* is usually invoked by other system, such as inventory management system or customer management system. After having the information of customer, the other interface *GenerateReceivableAccount(Account)* is invoked by payment management system to generate receivable account information. That is to say that the invoke sequence of these two interfaces are demanded to be complied in case of leading errors. This business constraint can be formally specified as the following temporal logic formulas.

Always Next(*GetAccountInformation(Customer), GenerateReceivableAccount(Account)*)

It means that once the action of invoking *GetAccountInformation(Customer)* is executed, the next action is always to invoke interface *GenerateReceivableAccount(Account)*, otherwise, this constraint is violated.

We have defined a language, called Business Property Specification Language (BPSL), based on LTL to specify business logic constraints. From implementation perspective, we propose to apply model checking technology to check the compliance of a composed service against all the business logic constraints involved in this composed service. The details of applying model checking technology into compliance checking have been introduced in our pervious paper [9].

Besides temporal logic rules, the intelligent service composer supports another two kinds of business logic constraints: *association binding* and *exclusion*. The association binding rules specify the constraint that multiple interfaces are always invoked associately. For example,

Always AND(*GetAccountInformation(Customer), GenerateReceivableAccount(Account)*)

Means that any one of the interfaces, *GetAccountInformation(Customer), GenerateReceivableAccount(Account)*, is invoked, then the other interface must be invoked in the same composed service. The exclusion rules specify that multiple interfaces should not be invoked at the same composed service. For example,

Exclusion(*GetAccountInformation(Customer), GenerateReceivableAccount(Account)*)

Means that any one of the interfaces, *GetAccountInformation(Customer), GenerateReceivableAccount(Account)*, is invoked, then the other interface should not be invoked in the same composed service.

SLA compliance: Besides the functional level constraints, each service component should provide SLA terms. As the previous introduce, the service components has include SLA related properties in their business specification file. However, in order to reach the predefined SLA goals, they may have some requirements on the composed components with it. For example, an interface *'GetAccountInformation(Customer)'* can provide 256 bit encryption service, however, the input data must also provide 256 bit encryption service while it is invoked by another services. So we suggest that the service provider should define the required SLA compliance for the to-be-composed components. The definition of the SLA compliance rules is similar to that of meta-information matching rules. For example, the security compliance rule example can be defined as:

Composite. Interface(*GetAccountInformation(Customer))*. INPUT. Security(encryption, protocol)=(256, SSL)

3.3. Template-based service adaptation

In section 3.1, we have introduced that the service components are organized with a well-defined *Service Catalog* to enable the quick searching**.** Service Catalog is a repository of service components from different service providers. It will be the general situation that some components can not be found from catalog. For this kind of situation, composer provides *service adapter templates* mechanism to help business man automatically generate adapter but not manual writing from scratch. While a required component is missed in the service catalog, the user can create an empty service which only has some business specification. The composed service with empty service can be saved as a service template in the user's personal workspace. When the new service is implemented, it can be used to replace the empty service in service template. And resave the service template as composed service in work space. This mechanism makes it flexible to replace a service template with different implementations.

In the real cases, users often meet the mismatched service components. If two service components come from different service providers, the common situation is that they are Partial Compatibility for composed service offering [6]. It means these two service components provide complementary functionality and could be linked together in principal; however, their interfaces and interaction patterns do not fit each other exactly. There are mainly four kinds of mismatch:

Signature Mismatch: it refers to the data that the service processing is different. Such as different data type, the number of data that can be processed per time.

Behavior/Protocol Mismatch: it refers to the message mismatch, such as different message order or sequence, different message name. etc.

QoS Mismatch: It refers to the non–function property mismatch, such as performance, max response time, etc.

Semantic Mismatch: Even the name is the same; it is possible that they have different semantics.

For QoS mismatch, we will include it in service compliance checking part. If the services do not compatible in QoS, then they are not allowed to be composed. For other situation, it is necessary to use mediators [6] to make the composition feasible. The basic approach is that we focus on mediator design time, set up mediator repository, define all kinds of mediator patterns and figure out mediator generating situations. For users' special requirement, composer will select the proper mediator, and for the mediator pattern that is not implemented yet, composer support users to create mediator template and implement it in later. During run time, mediator can generate automatically based on the situation match making.Due to the limited paper size, we will introduce this work in the separate paper.

4. UI-level composition

Different customers often have some specific requirements for UI; the intelligent service composer supports users to replace UI under the support of UI preview function and UI binding capability. Before introduce the details of UI composition, we first give the formal definition on some concepts mentioned in section 2.

Service component as the complete implementation of a business problem, including function, UI, and data implementation. We separate each part of service components as independent component that can be composed separately. Then the separate components include *service function component, UI component*, and *data component. Function Component* is defined as the function implementation of service component, including business logic and data operation implementation. *UI Component* encapsulate the implementation of the user interface of service component.

At UI composition layer of the composer, users can select and integrate UI components based on function composition result to enable flexible UI binding to the composed service. A service function component can bind a UI component or not when it is uploaded to the service catalog. At the same time, UI component can be developed independently and uploaded to the service catalog. We provide *UI preview function* to enable user to preview the UI pages and page flow before binding. And user can flexibly replace the UI components based on the specific requirements. We also provide personal workspace to save UI selection result.

5. Challenges of data-level composition

Users often face the problem that two components provided by different providers are composible from business function point of view but their data structures or data sources are different. If the data structures of two services don't matched, an intuitive way is to create mediators to transfer them. As we know, a lot of research works in this area can be leveraged by us. For example, [6]. Which describe the service composition problem from data angle, and give an example to show how to generate data mediator to make service compatiable. Another situation is that the data sources of two components are different, for example, one data source is xml style data definition, and the other data source is database. So it is necessary to generate data mediation to bridge them together. Some existing works or products such as schema matching database integration, etc. related to information federation or information integration

[11, 12] can be leveraged to federate different data sources.

Through the above analysis, we know that it is not a big problem to solve isomerous data structures or sources. However, to check the mismatching of two data structures is a big challenge for us because the semantics of data structure definition may be different. We can imagine that two data structures coming from different two providers may be different even they are completely the same from syntactical perspective. In fact, this problem has also been discussed in some papers [14, 15]. But there is no effective way to solve this problem. We believe an effective way to solve this problem is to require that all the service provides define their data structure based on the same standard, for example, Oagis [13] is a standard for retail industry data definition.

6. An example

In this section, we will go through an example to illustrate how to use composer to create a business application. The original real case is a big system, but we have simplified it as a small example here.

Example Background:

Client: ABC Clothes Company, this company is a small clothing company with about 200 employees; their main product is ABC sport suit.

Requirement: ABC Clothes Company intends to develop a sale management system, which supports customer to submit order and pay for the order online. In order to support online payment, this sales management system needs to integrate with an online payment system. Meanwhile, ABC Clothes Company can manage these orders and process correspondingly with the sales management system, such as check inventory, check out, customer information management, etc. ABC Clothes Company requires that the payment system can be replaced by other payment system flexibly but without stopping the system.

The related business people of ABC Clothes Company can use the intelligent service composer to quickly create a business application to meet the above requirement since a lot of service components have been available. Figure 2 gives the snapshot of web-based Intelligent Service Composer.

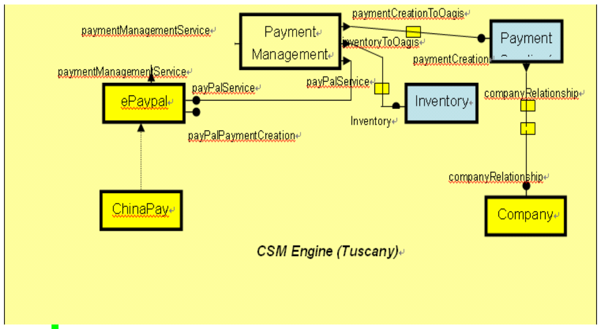


Figure 2: System Structure Overview

Overall, the complete business application development will go through two stages. In the engaging phase, only part of service components are available, then business people can quickly compose an incomplete application with available components and some service templates to quickly verify requirements. In the following producing phase, the missing service components are implemented and replace the service templates. In the following part, we introduce how to compose a business application with the intelligent service composer step by step.

In engaging phase

In the engaging phase, the appropriate service components are located from service catalog, and an initial application is created under the support of some advance features of the composer. After the composition, users can run a quick demo to verify whether the execution result can fulfill the requirement.

Step 1: selecting service components

Firstly, business people can find out a component 'Payment Creation' by searching some business keywords, such as 'payment' and 'retail'. Then the association searching capability of the intelligent searching can help quickly find out its associate component 'Company Relationship'. Figure 3 shows the snapshot of the intelligent searching. By following same approach, six components, 'Payment Management', 'Payment Creation', Company Relationship', 'Inventory Delivery', 'Paypal', and 'Chinapay', are searched out.

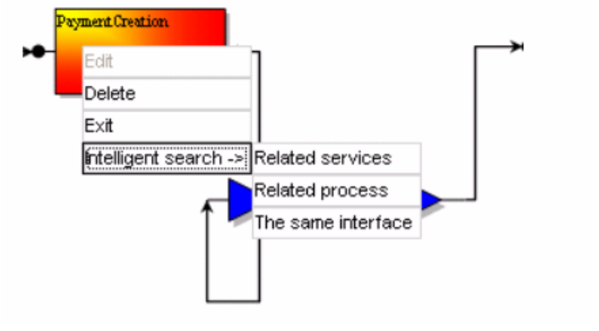


Figure 3: Intelligent searching

Step 2: Automatic wiring and mediators' generation

For those compatible service components, the **auto wiring** capability of composer can automatically check their consistency and connect them. If two components don't exactly match, the composer will firstly search whether their mediator has been provided in the service catalog. If no appropriate mediators can be found out, a mediator template will be generated, but the implementation of this mediator should replace the template finally. As showed in Figure 4, component 'Payment management' and 'Inventory Delivery' don't match, then a mediator is generated there.

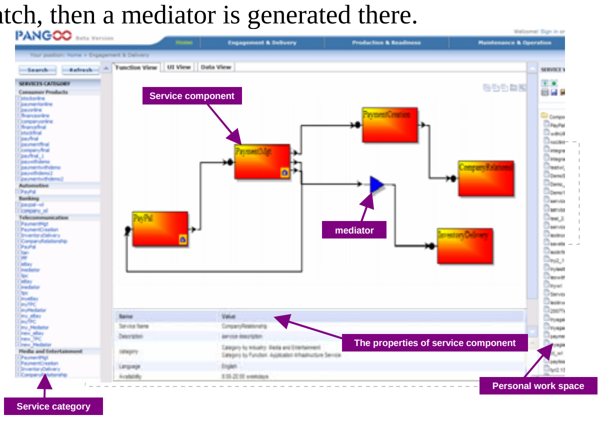
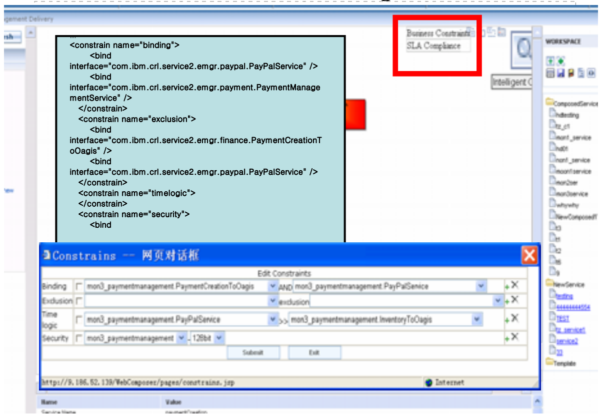


Figure 4: Composed service at functional level

Step 3: Service compliance checking

For checking the compliance of the composed service components, users have two choices to define compliance rules for this composed service. On the one hand, the embedded compliance rules of the composer can be selected, such as inclusion or exclusion rules; on the other hand, users can define some rules by themselves. In this case, a security related rules are defined which requires that the encryption byte of 'Paypal' component and 'Payment Management' component is the same (256 bit). The representation of the embedded compliance rules and user defined rules are showed in Figure 5.



```
<constrain name="binding">
      <bind
interface="com.ibm.crl.service2.emgr.paypal.PayPalService" />
      <bind
interface="com.ibm.crl.service2.emgr.payment.PaymentManage
mentService" />
   </constrain>
   <constrain name="exclusion">
      <bind
interface="com.ibm.crl.service2.emgr.finance.PaymentCreationT
oOagis" />
      <bind
interface="com.ibm.crl.service2.emgr.paypal.PayPalService" />
   </constrain>
   <constrain name="timelogic">
   </constrain>
   <constrain name="security">
      <bind
```

Constrains -- 网页对话框

Edit Constraints

Binding	mon3_paymentmanagement.PaymentCreationToOagis	AND	mon3_paymentmanagement.PayPalService
Exclusion		exclusion	
Time logic	mon3_paymentmanagement.PayPalService	>>	mon3_paymentmanagement.InventoryToOagis
Security	mon3_paymentmanagement	-	128bit

http://9.186.52.139/WebComposer/pages/constrains.jsp

Figure 5: Compliance rules

Step 4: Saving composed service in work space

The composed service can be save in user's personal workspace

Step 5: UI component binding

User can use UI preview capability to select proper UI for service offering. Service function component 'payment management' has two UI components, then the user can flexible bind different UI component with the service function component as showed in Figure 6.

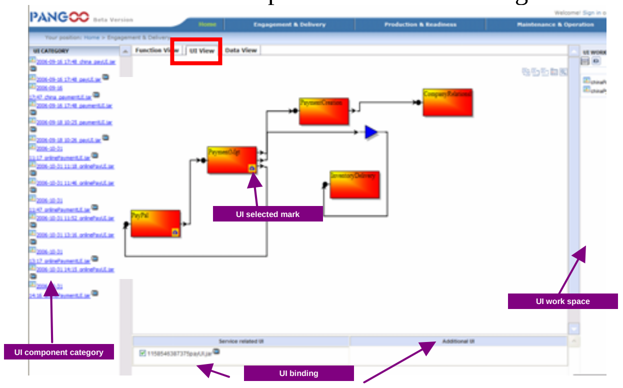


Figure 6: **UI binding**

Step 6: Data-level composition

In data composition view, data consistency checking tool can help check whether the data structure and source are consistent.

Step 7: Quick demo

The *quick demo* function can quickly deploy the composed service in the Tuscany platform of the composer, then the application can be executed. After looking at the execution result of the quick demo, ABC Company believes that this system satisfies their requirement. They will pay for the service components used in this composed service. All of above can be done through the intelligent service composer platform but without face to face communication, which will save a lot of engagement cost.

In producing phase:

During producing phase, the user needs to download the complete code of system to their own environment and run the application in their own environment. In addition, some templates need to be implemented and replaced.

7. Related works

There is a lot of work about web service composition. Here we list several main and most popular approaches for web service composition. *BPEL* [10] is an XML language that supports process oriented service composition developed by BEA, IBM, Microsoft, SAP, and Siebel. BPEL composition interacts with a Web services' subset to achieve a given a task. Another is *Semantic Web (OWL-S)* [1]. The Semantic Web vision is to make Web resources accessible by content as well as by keywords. The most important semantic web based composition tool is the Web Ontology Language for Services (www.daml.org/services). OWL-S (previously known as DAML-S) is a services ontology that enables automatic service discovery, invocation, composition, interoperation, and execution monitoring. Recently, *Web Components* [2] approach is getting more and more attention, IBM developed SCA (Service Component Architecture) as SOA programming model and support service composition. The Web component approach treats services as components in order to support basic software development principles such as reuse, specialization, and extension.

Besides above approaches, there are also some other approaches for web service composition such as algebraic process composition, Petri Nets method, model checking and Finite-State Machines, etc. But all these approached are developed from the developers' perspective and focus on the software application layer, but not from users' perspective or focus on business application level, and for business man it is hard for them to understand and do application from IT service view.

Some recent works in Web Service field aims to provide promising opportunities for integrating data, applications and business process. More and more works have discussed business level service composition or service integration from different enterprises. Such as *Framework for Semantic Web Process Composition* [3], which provides the enhancement of the current Web process composition techniques by using Semantic Process Templates to capture the semantic requirements of the business process. Another important work is *Role and Application of Pluggable Business Service Handlers in Web Services Choreographies* [4], which attempts to address business process integration issues, is Web Services Choreography Definition Language (WS-CDL)

A composition mechanism must therefore satisfy several requirements: Connectivity, Nonfunctional Quality-of-Service Properties, Correctness, and Scalability [5]. However, business level service composition is a very complex case of service composition, as it requires strong support for both business process understanding and technical infrastructure developing in order to satisfy the connectivity at all levels. So we put forward a unified service composition framework which covers 3-layer composition. With this composition framework the connectivity of composition at all levels can be fulfilled.

8. Conclusion and future works

We present a service composition framework which covers 3-layer composition, including functional composition, UI composition, and data composition. An intelligent service composer based on this framework enables business people to compose business application from a set of service components. The service composer provides three important advanced capabilities, intelligent service searching, automatic compliance checking, and template-based service adaptation, which enable non-technical people quickly locate the required service components, quickly check the compliance of these components, and automatically connect these components. The web-based composer allows business people to try the function of the composed service by using the quick demo function. We have applied this composer into a service delivery platform which supports business people, such as end users or business partners, to compose business applications by themselves.

In fact, we only give the overall introduction of the service composition framework in this paper but without deep technologies introduction. As we have mentioned in the previous sections, there are lot of technical challenges in order to enable business people to develop business applications, such as data adaptation and function interface mediation. In future, we will have deep research on the consistency checking of two service component from function, UI and data perspectives.

9. References

[1] A. Ankolekar et al., "*DAML-S: Web Service Description for the Semantic Web,*" Proc. Int'l Semantic Web Conf. (ISWC), LNCS 2342, Springer-Verlag, 2002, pp. 348–363.
[2] J. Yang and M.P. Papazoglou, "*Web Component: A Substrate for Web Service Reuse and Composition,*" Proc. 14th Conf. Advanced Information Systems Eng. (CAiSE 02), LNCS 2348, Springer-Verlag, 2002, pp. 21–36.
[3] Kaarthik Sivashanmugam, John A. Miller, Amit P. Sheth, Kunal Verma. "*Framework for Semantic Web Process Composition*" Large Scale Distributed Information Systems (LSDIS) Lab, Computer Science Department.
[4] Svirskas, A. Wilson, M. Roberts, B. Vilnius Univ., Lithuania, "*Role and application of pluggable business*

service handlers in Web services choreographies" Data base and Information Systems, 2006 7th International Baltic Conference Publication Date: 3-6 July 2006 On page(s): 194- 201ISSN: ISBN: 1-4244-0345-6

[5] Nikola Milanovic and Miroslaw Malek, *"Current Solutions for Web Service Composition"* (spotlight)

[6] Wei Tan. , Zhong Tian. , Fangyan Rao., Li Wang., Ru Fang, "*Process Guided Service Composition in Building SoA Solutions: A Data Driven Approach*". Web service, 2006. ICWS'06. International Conference
Publication Date: Sept. 2006 On page(s): 558-568 Location: Chicago, IL, USA,ISBN: 0-7695-2669-1

[7] Thomas Erl, *"Service-Oriented Architecture (SOA): Concepts, Technology, and Design."* The Prentice Hall Service-Oriented Computing Series, ISBN 0-13-185858-0.

[8] *Service Component Architecture Specifications*, http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications.

[9] Y. Liu, S. Müller, K. Xu, *"A Static Compliance Checking Framework for Business Process Models"*. Special Issue on Compliance Management, IBM Systems Journal, 46(2). (2007).

[10]*Business Process Execution Language.* Http://www.ibm.com/developerworks/library/ws-bpel.

[11] Philip Bohannonz Wenfei Fany _ Michael Flasterz P. P. S. Narayan, "*Information Preserving XML Schema Embedding"*

[12] Zhen Zhang Bin He Kevin Chen-Chuan Chang
"Light-weight Domain-based Form Assistant: QueryingWeb Databases On the Fly"

[13]*"OAGIS 9.0 Introduction", http://xml.coverpages.org/OAGISv90-Introduction20050809.pdf*

[14] *Bringing Semantics to Web Services: The OWL-S Approach*, Book Series Lecture Notes in Computer Science, *Semantic Web Services and Web Process Composition*, Springer Berlin / Heidelberg , ISSN 0302-9743 (Print) 1611-3349 (Online) , Volume 3387/2005, ISBN 978-3-540-24328-1, Pages 26-42.

[15] Foster, H. Uchitel, S. Magee, J. Kramer, J., *Model-based verification of Web service compositions*, appears in: Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on Publication. Page 152-161, ISSN: 1527-1366, ISBN: 0-7695-2035-9.