Dealing with Web Services Composition at the Architechtural Level

Djamal Bennouar
Kahdidja Bentlemsan

Dealing With Web Services Composition at the Architectural Level

Djamal BENNOUAR
Computer Science Department
The Saad Dahlab University
Blida, Algeria
dbennouar@ens-kouba.dz

Walid Khaled Hidouci
The National School of Computer Science (ESI)
Oued Smar, Algiers, Algeria
w_hidouci@esi.dz

Kahdidja Bentlemsan
The National School of Computer Science (ESI)
Oued Smar, Algiers, Algeria
bentlemsan_khadidja@yahoo.fr

***Abstract* — Design by Component composition using connectors as a glue to produce software system represents the fundamental practice in software Architecture. Software Architecture approaches use a component model which explicitly exposes through its interfaces the needed and the provided resources. Component in Software Architecture are deployable unit of composition. They may be instantiated many times and deployed in different containers. Web Services and services in general, are usually already deployed software. They may correspond to third party services which cannot be touched in any way. The Integrate Approach to Software Architecture (IASA), originally defined for hard soft co-design, provides a number of concepts, not present in current Software Architecture approaches, which may use to efficiently deal with the service concept and the web service composition problem. IASA not only provides a simple and efficient solution to the web service composition problem, but enables the definition of aspect oriented heterogeneous system where some parts are handled by web services technology and other parts are handled by component deployed in any other software technology. Defining heterogeneous composition is not supported by current web services composition languages.**

Keywords: ***Web Service, Software Architecture, IASA, Composition, Component, Orchestration.***

I. INTRODUCTION

Today, the use of web services has become a common practice in software industry for supporting both Business-to-Consumer (B2C) interaction and Business-to-Business (B2B) collaboration. Through Web service composition, new complex web services may be created by reusing existing ones. Several organizations have proposed a number of languages oriented to deal with the Web Service composition problem such as XLANG, WSFL, BPML, BPMN, BPEL4WS, BPSS, WSCI, and WSCL.

In practice these languages requires a hard work and the software designer must have high skills to correctly construct a composite web service. Experiences conducted by [1] and [2] has shown how is difficult the design and the realization of small part of a complex system using web services composition language even in the context of an advanced IDE.

As an example, BPEL4WS [3, 4], which seems to be the most used language, is not only very complex to learn but hard to be operated. This complexity is due to the low level of expressivity. The software designer must deeply master the structure of the BPEL file, i.e. how and where to use the appropriate BPEL activity.

In addition we have found that the composition languages of web services are limited to the construction of complex web services, and cannot provide heterogeneous composition of ordinary components and web services or an aspect oriented composition.

The work presented in this paper presents a new approach to web service composition problem. This approach not only reduces the composition complexity but allows the heterogeneous composition of any component with web service. The main idea dealing with the web service composition problem is to bring the composition problem from the implementation level represented by web services composition languages, to the architectural level. At the architectural level the composition problem is considered at a high level of abstraction totally independent from any implementation technology.

In the work presented here, we use the Integrated Approach to Software Architecture (IASA)[5] which offers a high degree of freedom from any software mechanism constrain when specifying a composite component. In IASA we can see web services as IASA components. Connection between web services are then achieved using IASA connectors.

In the remaining of this paper, we will first introduce the IASA fundamental model elements used to deal with web service composition. Then we will show through a case study how IASA can improve the objectives fixed above

II. THE IASA APPROACH

In IASA, an application is a composite component type made of instances of other component types which may be composite or primitive component types. The specification of an application is achieved through the IASA Architecture Description Language (ADL) called 3ADL (Architecture, Action and Aspect Description Language)[6]. The concept of action, which is the 3ADL base for describing the miscellaneous behaviors in IASA, is taken form *Precise Action Semantic* of UML [7].

A composite component type has a specific internal view organization which consists of three parts: the *operative part*, the *control part and the aspect part* (figure 1). The first contains pure business components. The aspect parts contains component dealing with technical concerns needed to efficiently operate the business part such as security, logging, and persistence. The control part is made of a number of controllers which are components dedicated to control the whole composition (i.e. operative part initialization, dynamic aspect injection, dynamic architecture evolution etc...). At least one component, named *main*, must be instantiated as a controller. The *main* controller is a behavioral component which means that it is specified using the action concept of the 3ADL *language*[6,8].

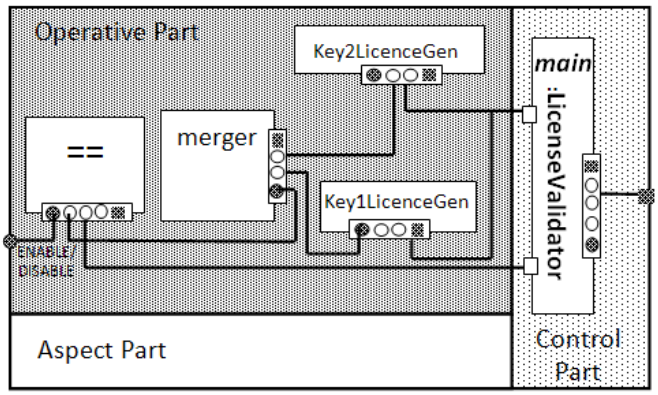

Figure 1. The Internal View of a Composite Component

A component type is instantiated using the envelope concept. An envelope is a kind of wrapper [9] which is a well-known technique used to deal with the architectural mismatch problem[10] and to isolate component's incorrect behaviors from the rest of the system in order to allow nonstop computing. In addition to the usual objectives of wrappers, the IASA envelope is used to specify the deployment map, to enable the specification of connections involving the port's structural elements and to manage the injection and deletion of the advices provided by aspect components[11].

The deployment map specified as part of an envelope describes the deployment environment or container, the deployment case of the component in such environment and the preferred implementation technology when the deployment case is not associated with a specific one. The deployment environment or container may be a machine name with its operating system, an application server or an already deployed component. The deployment case describes the exact nature of a component instance in its deployment environment. As an example, *PROCESS*, *EJB* (Enterprise Java Bean), *THREAD* are three supported deployment cases. The first is valid in a deployment environment represented by a machine name and its operating system. The second is valid in a deployment environment represented by a J2EE compatible application server. The third is valid for a deployment environment represented by a component already deployed as a *PROCESS*. The preferred implementation technology is usually a programming language. When a deployment case is specific for a particular implementation technology, like the *SERVLET* or *EJB* deployment cases which are specific to the Java language, the deployment technology is implicit in a deployment map.

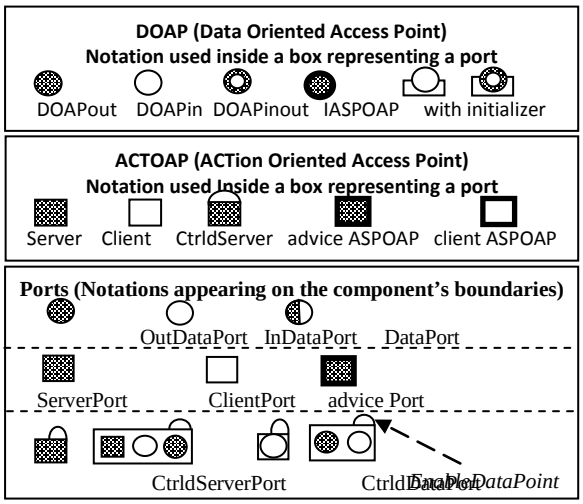

Figure 2. Main IASA Graphic Notations

A IASA primitive component is totally independent from the context where it is used. The adaptation of a primitive component to its execution environment is achieved through the envelope. In current IASA version which is mainly based on the java technology, a primitive component is represented by a POJO[1] class. As an example, deploying a primitive component as a web service will produce an envelope provided with necessary resources needed to view this pure primitive component as a web service. The production of this implementation view of the envelope is achieved by the IASA transformation process which starts from a 3ADL description and produce the implementation view in the chosen implementation technology.

The Link Component (LComponent) is a concept introduced by IASA [11] to explicitly show the interaction between the components of a software system with the external components such as the execution environment, the file system, servers etc. Such interaction is usually ignored in current software architecture approaches. The LComponent is also used to represent the concept of shared component.

Like any other component instance, the LComponent is associated with an envelope which enable to view the same external or shared component from a different perspective. The LComponent and the envelope concept are fundamental in the composition process of web services according to IASA.

A IASA component interacts with the external world through a set of port. A IASA port is made of access points. As opposite the other approaches, the access points may be manipulated individually or in group by the IASA connectors[11]. A IASA access point is either a Data Oriented Access point (DOAP) or an Action Oriented Access Point (ACTOAP). An ACTOAP is either a client or a server. It is associated with a service represented by a number of actions.

A DOAP is always associated with a specific data type. Any DOAP is associated with a number of predefined

[1] Plain Old Java Object

actions enabling data communication and reporting miscellaneous events (i.e. *send receive, open, close, accessed, updated, changed*). The main graphic notation of IASA ports and access points are shown in figure 2.

A port maintains an abstract view and a concrete view. The abstract view is represented by the concept of access point, the actions associated with an access point and the port's behavior. This later is represented by a set of valid rules using the concept of action of the 3ADL language. Each rule shows how the required or provided resource must be used. The concrete view may be any model, provided with a clear way leading to the implementation level.

The Current implementation of IASA is based on a number of specific ports: the *ClientPort,* the *ServerPort,* the *AdvicePort* and the *DataPort.* The *ClientPort* is made of a single client ACTOAP and a number of DOAP. The *ServerPort* uses one server ACTOAP and a number of DOAP. The *DataPort* is made of DOAPs. The *AdvicePort*[11] is a *ServerPort* provided with an Aspect Oriented ACTOAP server called ASPOAP instead of a server ACTOAP. The concept of *advice ClientPort* is not supported, since it strongly reduces the obliviousness concept of a component [12].

The IASA connector model is based on a behavioral view and a structural view. The behavioral view describes an interaction and the structural view defines the infrastructure needed to transport the interaction (e.g. procedure call, RPC, FTP, HTTP, SOAP etc..). A valid interaction, described using the 3ADL action concept, does not violate the behaviors of the interconnected ports.

III. WEB SERVICE COMPOSITION IN IASA

In IASA, the web services concept is considered as a concept belonging to the implementation level view. Hence, it must be described by a deployment case. At the architectural level, any component instance is manipulated independently from its deployment nature. The same architecture may be deployed totally as a composite web services or as an heterogeneous composition which is a mixture of web services and other services or completely without web services. Moreover, in the same composition, an instance of a service, represented by a component type, may be deployed as a web service and another instance of the same component type may be deployed as a non web service.

To be able to deploy a component instance as a web service, we have introduced in IASA a new deployment case called JWS (Java Web Services). Like other deployment case (EJB, SERVLET, JSP, BEAN, CLASS) the JWS is based on the Java Language[2]

Theoretically any IASA component may be deployed in any of the cited implementation technologies. Hence it is possible in IASA to have in a composition many instances of the same component type and deploy one instance as an EJB, another as a SERVLET and a third as as a Web Service etc. All such instances use the same POJO kernel, but have different envelope.

[2] Currently Java Represents the Unique Language Supported by IASA

The IASA approach make use of a number of concepts to achieve the composition of web services without any impact on the concepts and standards associated with web services and Service Oriented Architecture (SOA) in general (table 1). In the general situation, where services are non deployable unit of composition, IASA uses the LComponent with the envelope provided with the JWS deployment case. The LComponent represent an already deployed web service. The envelope associated with each LComponent representing a web services will be deployed with the controllers (component of the control part). The controllers and the envelopes will encompass the composition logic which may be an orchestrated logic, a choreography logic or any other logic supported by the IASA approach .

When a web service is handled by a deployable component, not an LComponent, the envelope is deployed with the component instance. The deployment of an internal new web service may be followed by the registration of its WSDL descriptor in a specific web services registry.

TABLE 1: IASA Mechanism to Achieve WS Composition

SOA and Web Services	IASA Mechanisms
A third party service or a third party web service	LComponent
Deployable web service	Component
Service Interface or WSDL	Port Structure
Service Interface Behavior or WSCI	3ADL Port behavior
Web Service Orchestration	Controller and Envelope of LComponent
Web Service Choreography (WSCL)	3ADL controllers behavior, port behavior, connector interaction (Abstrcat)
Executable choreography	Controllers and Envelope of LComponent

The main difference between IASA web service composition and current web service composition languages such as BPEL, is represented by the fact that IASA generates a number of POJOs handling the orchestration or choreography logic and the envelopes needed to transform each generated POJOs to a web service. Deploying IASA composite web services does not require the installation of a specific and heavy orchestrator engine as this is the case when the composition is achieved using a web service composition languages.

IV. THE TRANSFORMATION PROCESS

One of the main important parts of the IASA approach is represented by its transformation process subsystem [5]. This later takes a 3ADL abstract description and produces an implementation view in the selected implementation technologies. The transformation process is made of steps independent from any implementation technology and steps specific for each implementation technology, even if these technologies uses the same programming language. Current policy in the IASA project is

to deal in an independent manner with each implementation technology. The planned objective is to build a repository of transformation rules which may be used later as a knowledge base in the context of future research dealing with the enhancement and optimization of the IASA transformation process.

The current process of generating the implementation view in IASA depends on the specified topology, the communication mode of the access points, the chosen implementation technology for each component instance, the deployment case of each component instance and the used standards for ports and connectors. The transformation process consists of the three following steps:

- **Aspect Weaving:** This step, achieved at the envelope level, modifies the structure and the behavior of a envelope port and produces an intermediate weaved 3ADL description.
- **The Normalization Step:** This is the fundamental step in the process. It is based on a number of transformation rules and produces a regular architecture where IASA ports are transformed into ports fully provided with the required software mechanism (e.g. an IDL description, a Java Interface description, an *ArchJava* port, UML).
- **The Production of the Implementation View:** This third step is mainly based on the deployment map. It provides the ports with the necessary adapters, solves the distance problem between connected component and attaches the port to a connector endpoint

The deployment of a component as a web service is achieved at the third step. Two great actions are realized in this step for producing a composite web service:

- The production of the POJOs representing the behavior of the controllers described in 3ADL
- The execution of a number of rules to generate the implementation view of the envelopes of all components involved in the composition (operative part component, aspect part component and control part components).

Here is some specific rules used to deploy an instance of a component type as a JWS.

- A web service envelope is composed of a POJO, the stub of the required web services, the WSDLs of the required web services and the WSDLs of the provided web services.
- The envelope first role is the localization of the needed web services in order to get the necessary stub and WSDL files.
- The envelope POJO must implement the same interface as the wrapped component which is usually represented by a POJO.
- The envelope POJO must be provided by the reference of the wrapped component. This operation is achieved at the envelope POJO constructor.

The following code shows the POJO representing a component type called *AdderCmp*, the interface implemented by *AdderCmp* and the envelope POJO with necessary java annotations. The java annotations are used to automatically generate the WSDL file describing the AdderCmp web service[3] which is here a deployable web service.

```
package iasa.component.javaws.primitive;
public class AdderCmp implements AdderPort{
        public int add(int a, int b){
                return a + b;
        }
}
```

```
package iasa.component.javaws.primitive;
import iasa.ports.*;
public interface AdderPort extends IASAServerPort {
        public int add(int a, int b);
}
```

```
package iasa.impl.ws.envelope;
import iasa.component.javaws.primitive.*;
import javax.jws.WebMethod;
import javax.jws.WebService;

@WebService
public class AdderWSEnvelope {
        AdderPort adder;
        AdderWSEnvelope (){adder = new AdderCmp();}

@WebMethod(operationName="add" )
   public int add(int a, int b){
        return adder.add(a, b);
   }
}
```

V. CASE STUDY

The objective of this case study is to show the usefulness of our approach. First we design a new composite component using the IASA top down design process [5]. This composite component implements the following mathematical function.

$$f(x, y, z) = (x + y) * z^2$$

We consider that all components are external components (this is usually the case when composing web services). The three already deployed components are represented by an LComponent in the design. The composite component named FCmp is made using LCAdderCmp, LCMultiplierCmp and LCSquareCmp (figure 3).

[3] Current JWS deployment case is based on the JAX-WS 2.0 API

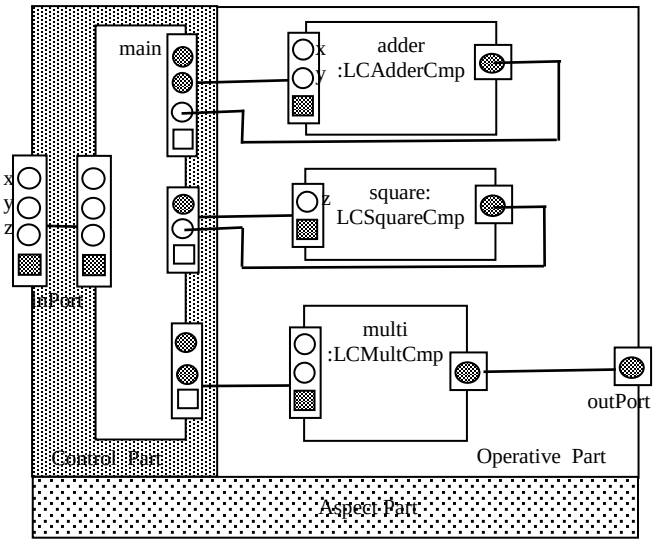

Figure 3. FCmp Architecture Based on an Orchestrated Composition Logic

The FCmp component has two ports. The first one, named *inPort*, is a service port containing a server ACTOAP and three DOAP. The server ACTOAP enables the activation of the f service. The three DOAP handles the x, y and z operands of f. The second port, named *outPort*, is a data port containing only one DOAP used to return the result of the f service.

The control part is made of one controller. One of the possible behaviors of the *main* controller, written in x3ADL (the XML version of the 3ADL language), is described in figure 4. This behavior states that all the services provided by already deployed web services are launched in sequence, in a synchronous mode. First the adder is invoked and its result is captured by the controller. Then, the square is launched and finally the controller launches the multiplier which produces the final result.

```
<fire   component = “adder”    service= ”add”
        result = “out” mode = ‘synchrone”/>
<fire   component = “square”    service= ”square”
        result = “out” mode = ‘synchrone”/>
<fire   component = “multi”    service= ”multiply”
        result = “out” mode = ‘synchrone”/>
```

Figure 4. Partial view of the *main* controller behavior specified using x3ADL

The deployment specification reported in figure 4 first describes the environment where the component may be deployed and the supported deployment cases. The deployment map indicates the effective deployment of all the component instances realizing the composition. Usually the first specification targets the main controller which represents the starting point of the component execution. Figure 5 shows two possible orientations when the deployment target LComponent. When an LComponent is targeted by the deployment, the directive in applied only to the envelope since the LComponent is an already deployed component.

In the first orientation, represented by the directive “**deploy rall in** jboss”, all the envelope (which are actually POJOs provided which necessary stub and skeleton) are deployed in distinct web service. Each envelope plays the role of a proxy to the web service specified in the composition. The composition is here represented by a kind of multithreaded application where each thread is a web service by itself. This Orientation is suitable for a choreography composition.

In the second orientation, represented by the directive “**deploy rall in this**”, all the POJOs envelope are inserted in the *main* controller, yielding only one web service. This kind of deployment seems to be more suitable for orchestrated composition.

The two orientation may be used simultaneously. As an example, the envelope of *adder* (figure 4) may be deployed standalone and the envelope of *square* and *multi*(figure 4) may be deployed as part of the *main* controller.

```
////// File FCmp.dpy
// Deployment architecture, GCCF and cases
package fcmp.component
component FCmp {  // indicates that the following declaration
  architecture {       // are part  of the FCmp component type
      environment jboss {
        machine  localhost;
        container jboss5.0 ; //JBoss Server 5.0
        namespace  fcmp ;  //localhost:8080/fcmp
       os UNIX; // Generic name used.
      }
      deploymentcase {JWS}
 }
// Deployment map definition Many maps may be defined  for the
// same application
deploymentmap ws_on_jboss5 {
   // First specify the main controller deployment
   deploy this as JWS in jboss;
   // Now specify the deployment for all other component
   // The next line indicates that each envelope will be deployed as a
   // standalone web service. Recommended for orchestrated
  // deploy rall as JWS in  jboss ; //  rall : Recursive Deployment:
                                    //  target the composition tree
  //uncomment the following deploment directive
  //  if you want to represent th ecomoosition by a single
  // web service (recommended for orchestration), we deploy all
  // other envelope in the main controller
deploy rall in this;
}
```

Figure 5. Deploying FCmp Composite Component as a Composite Web Service

VI. CONCLUSION

In this paper, we presented the web service composition problem solved at the architectural level with the IASA approach. Regarding the composition using Web Service Composition Languages, the IASA approach allows the designer to compose web services in the same way they compose any other services, by the use of a component model and connectors. The designers are then insulated from dealing with the complexity of the implementation level represented either by programming languages or web service composition languages. The main difference between the IASA approach and the current web service composition languages is located in the representation of the composite web service. While the IASA approach produces one or more standalone and lightweight web service based on a POJO, a web service

composition languages approaches usually uses a heavy engine to execute the composition represented by an XML description. This heavy engine must be installed in the application server prior to deploy any composite web service.

The conducted work has shown some challenges the IASA approach is currently facing. The main challenge is represented by the huge code generated by the transformation process from an abstract view to an implementation view. The huge code is due to the envelope concept. Currently, for each component type instance an envelope is generated. This situation leads to the generation of redundant information such as WSDL descriptors, stub and skeleton. Optimizing redundant information, while generating the implementation level code, represents one of the most important actions in the future works of the IASA approach.

REFERENCES:

[1] K. Bentlemsan, "Composing Web Services Using A Software Architecture Approach", Magister Thesis, Computer Science Department, The Saad Dalhab University at Blida, Algeria, (2010).

[2] D. Bennouar, H. Benasmane, "Evaluation of the Web services composition approaches using web services Composition Languages in the Context of the Design of the eAPC eGovernment System", *Internal Report* N° IR.IASA.LRDSI.CS.2010.04, LRDSI lab, The Saad Dalhab University at Blida, Algeria, (2010).

[3] M. Keen, J. Cavell, S. Hill, C. K. Kee, W. Neave, B. Rumph, H. Tran, "BPEL4WS Business Processes with WebSphere, Business Integration :Understanding, Modeling, Migrating", IBM Redbook, (2004)

[4] M. Patino, P. Jimenez, S. Perez, "A Visual Web Service Composition Tool for BPEL4WS", *Visual Languages and Human-Centric Computing, IEEE Symposium.* Page(s): 181 – 188 (2005)

[5] D. Bennouar, "The Integrated Approach to Software Architecture", Phd thesis, ESI, Oued Smar, Algiers, (2009)

[6] D. Bennouar, A. Henni, "A Review of an Aspect Oriented Architecture Description Language", *The Mediterranean Journal of Computers and Networks*, Vol. 6, No. 1, (2010)

[7] OMG, "Action Semantics for the UML, Final Submission", TR, Object Management Group, 2001

[8] A. Saadi, "An Action Language for the Specification and Validation of Software Architecture Behavior", Magister Thesis, LRDSI Lab, Computer Science Department, The Saad Dahlab University, Blida, Algeria, June 2008.

[9] R. DeLine, "A Catalog of Techniques for Resolving Packing Mismatch". *Proceedings of the 5th Symposium on Software Reusability*, Los Angeles, 1999, p44-53.

[10] D. Garlan, R. Allen, J. Ockerbloom, "Architectural Mismatch: Why Rreuse is So Hard", *IEEE Software, 12(6):*17--26, Nov. 1995

[11] D. Bennouar, T. Khammaci, and A. Henni, "A New Approach for Component's Port Modeling in Software Architecture", *Journal of System and Software*, doi:10.1016/j.jss.2010.03.005, Elsevier, 2010.

[12] R.E. Filman, D.P. Friedman, "Aspect-Oriented Programming is Quantification and Obliviousness", Workshop on Advanced Separation of Concerns, OOPSLA 2000.