Migrating Component-Based Web Applications to Web Services: Towards Considering a "Web Interface as a Service"

Chouki Tibermacine, Mohamed Lamine Kerdoudi

Migrating Component-based Web Applications to Web Services: towards considering a "Web Interface as a Service"

Chouki Tibermacine
LIRMM, CNRS and
Montpellier II University, France
Chouki.Tibermacine@lirmm.fr

Mohamed Lamine Kerdoudi
Computer Science Department
University of Biskra, Algeria
lamine.kerdoudi@gmail.com

***Abstract*—Web component-based development is a challenging development paradigm, whose attraction to practitioners is increasing more and more. One of the main advantages of this paradigm is the ability to build customizable and composable web application modules as independent units of development, and to share them with other developers by publishing them in libraries as COTS (Commercial Off The Shelf) or free components. In parallel, since many years, Web services confirmed their status of one of the most pertinent solutions for a service provider, like Google or Amazon, to open its solutions for third party development. In this paper, we present an approach to migrate existing web component-based applications to a set of primitive and composite Web services and deploy them on a web service provider. This transformation helps server-side web application developers in transforming their "user interface"-based web components into a set of web services intended for remote code extensions. We implemented our solution on a collection of Java-related technologies. Java EE components are the input of the proposed implementation, and a set of Java Web services with their WSDL interfaces, choreographies and orchestrations of these services are provided at output.**

***Keywords*-Web Component, Web Service, Code Migration and Java EE**

I. Introduction: Context and Motivation

Since the end of the nineties, Web component-based development has emerged as a new solution which aims at decoupling Web application code modules, and making them reusable and customizable software entities. Indeed, a step has been taken forward in modularizing Web applications and thus separating business logic code, from view, data model and operational control one. One of the technologies leading this field is Java EE and its numerous frameworks like Struts or JSF. Web modules in such technologies are entities that can be used and reused in different applications and customized according to the application requirements. Many libraries in the Internet provide access to Web COTS (Commercial Off The Shelf) like ComponentSource ®[1] or free Web components like RichFaces from JBoss or Apache's MyFaces.

These technologies are currently one of the most interesting solutions for developing large and complex applications with highly critical requirements on maintainability and portability. However, after deploying a Web component-based application within an application server, there is no means to directly publish some services of the application for third party development. Even if stubs can be generated and provided for client applications, these stubs are generally language-dependent (only Java clients can use stubs generated for EJBs) and cannot be published, as they are, in libraries of services. Recently, the EJB 3.x specification introduced some annotations to enable developers to publish some methods in a bean as services. However, this is possible only for individual methods, and not for compositions of them which we found in real-world business logic. In addition, this solution is well suited for new software, but not for legacy one, whose source code is not always accessible. The same observations can be made on Eclipse tools (WTP project), which allow to generate Web services starting from individual methods in Java classes.

Besides, since many years, Web services have confirmed their status of one of the most pertinent solutions for a given service provider, like eBay (auction and shopping), Amazon (retail) or FedEx (logistics), to open their solutions for third party software development. Web services are functionalities based on standards which are "programming language"- and "execution platform"-independent, like WSDL or SOAP. New applications with thin or thick clients can be built and can access these functionalities, by simply formulating requests, which embed XML-based (SOAP) messages, to the chosen Web service providers. The same kind of messages are returned back to these applications, containing the results (answers to their requests). Upon these results, more actions can be performed by these new applications in order to implement some new business-logic.

In this paper, we present an approach (Section III) to build Web service-oriented architectures starting from existing Web component-based applications and deploy them on a Web service provider. In this way, developers of Web components can offer the opportunity to other developers to build extensions of the services provided by their artifacts. This transformation goes through several steps, starting from the parsing of Web components' contents and formatting

[1]ComponentSource Website: http://www.componentsource.com/index.html

Web interfaces as operations embedded in services, and compositions of these services, and ending by deploying and indexing the generated services.

We implemented our solution (Section IV) on a collection of Java-related technologies. Java EE components are the input artifacts of the proposed implementation, and a set of Java Web services and choreographies/orchestrations of these services are provided at output. We illustrate this approach on a simple example of a simulated version of a real-world Web application (Section V). Before concluding and presenting the ongoing and future work at the end of this paper, we make a summary of the related work (Section VI). In the next section, we introduce a simple example of a Web component that is used in the remaining of the paper to illustrate our proposals.

II. Illustrative Example

We consider here a simple Web component, which is composed of the following subcomponents:

- a `BMI_Calculator` calculates the Body Mass Index (BMI) starting from the values entered by the user via the Web interface implemented in this component. These values are the mass and the height of the person[2].
- a `DietaryAdvisor` asks the user to enter her/his age and gender, and to check some boxes representing the dietary habits and the eating disorders of the person (excessive meat eating, mid-night hungers, ...). It provides a summary of the dietary guidelines.
- an `EMailer` asks the user to enter her/his e-mail address and sends her/him an e-mail with a detailed list of dietary guidelines.

The Web component in this example is a Java EE EAR (Enterprise ARchive) which contains some Web and EJB modules. The Web modules are implemented using the JSF (JavaServer Faces) framework, which binds the input values got from the HTML forms (rendered by some JSF components) to JavaBeans. The EJB modules connect to a database via JPA (Java Persistence API) to get the stored information about the dietary guidelines. They also use javax.mail to send e-mails via an SMTP server.

Suppose now that a third-tier developer would like to implement a more sophisticated application based on the component services introduced above. This developer would like to provide a solution to sell diet nutrition and dietary recipes. She/he wants to use the output given by the `DietatyAdvisor` to advertise her/his products before sending the e-mail to the user.

In order to implement this solution, the third-tier developer should either: i) re-develop the business logic implemented in the Web components introduced above, ii) ask the provider of these components for the binaries (or source packages of these components) and deploy them on the same name space of her/his solutions, iii) or ask for a remote access to these components. It is evident that the first alternative is not realistic for this developer (a time consuming task). The other two alternatives are not good choices neither. The original developers of the Web components would certainly not give access to their software artifacts (binaries or sources) for many technical reasons (security, data inconsistency, etc.).

One of the best solutions for this third-party development is to transform the previous Web component-based architecture to obtain a set of Web services which are described below[3]:

- `DietaryAdvising`: a Web service composed of three operations :
  - `calculateBMI` which receives as input two messages of type Integer (the height in centimeters and the mass in grams) and returns a float value which represents the BMI.
  - `getBMI_Category` which receives as input the BMI value and returns the BMI category: Underweight, Normal Weight, Overweight, or Obesity.
  - `provideSummaryDG` which receives as input four messages: a message of type integer for the age, a boolean message for the gender, a float one for the BMI and a complex XML schema type for the dietary habits. This operation returns a summary of the dietary guidelines.
- `MailSending`: a Web service with a single operation for sending an e-mail. This operation declares four messages: three input strings for the e-mail address, the subject and the body of the message, and a boolean output message which represents the message sending success or failure.

As stated in the introduction, since many years Web services have confirmed their status of highly secure, portable and flexible solutions for distributed service computing. In this way, the application extension scenario introduced above can easily be implemented by remotely invoking the Web service operations. The extension of the Healthy Diet application can be built as a BPEL (Business Process Execution Language) process invoking successively the three operations of the first service. It then adds the corresponding advertisement information, and at last invokes the `MailSending` service.

The transformation process which allows the generation of the Web services above is presented in detail in the next section. We explain, among other issues, how the different operations in Web components are extracted and grouped in Web services. In addition, we show how elements in Web interfaces are formatted as SOAP messages, and how

[2] $BMI = \frac{mass(kg)}{height^2(m^2)}$

[3] This is not an exhaustive list. Some other generated operations will be introduced in the following section.

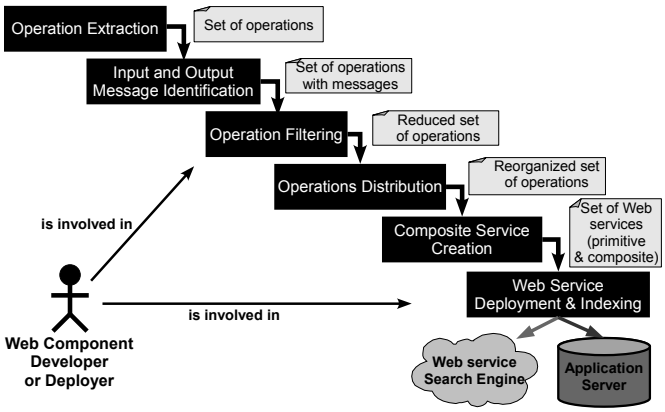

Figure 1. The Transformation Process

operations are created for representing these Web interfaces, and included in the published Web services.

III. Proposed Approach

The transformation of Web components into Web services is a six-step process. This is illustrated in Figure 1 and each step is detailed in the following subsections.

A. Operation Extraction

First, a recursive parsing of the different Web component elements is performed to extract the potential set of Web services. All operations in classes and other structured code elements are saved. In addition, the code present in programs executed at the server side (JSP pages, for example) is grouped within new operations and formatted to be executed as stand-alone code. For example, all code present in scriptlets of JSP pages is grouped and formated within a single operation. In the next subsection, we explain how this formatting is performed.

B. Input and Output Message Identification

The input and output messages related to each operation in Web services are extracted starting from the parsed elements in the Web components. For operations in classes and other structured code elements, the parameters and the returned values are formatted as (respectively, input and output) SOAP messages. The code present in the other programs (like JSP pages) is parsed to extract the input values received in the HTTP requests (by identifying statements getting values from HTTP requests). Their types are deduced from the parsed code by analyzing type casts and other conversion statements. The contents produced by theses programs, which are viewed at the client side (like JSP expressions or out.println(...) statments), are considered as returned values. The types of these values are extracted from the code and defined in the generated SOAP messages.

For example, in the `BMI_Calculator` component of the previous section, two messages are defined starting from the `request.getParameter(...)` statements in the JSP file. They correspond to the mass and height of a person. At the beginning, these inputs are defined as variables of type String. But, after parsing the JSP (Java) code that extracts these values, we deduce the final data type, which is int (integer). Nevertheless, the conversion, the cast and the `request.getParameter(...)` statements will be deleted from the new code. The returned value is the result calculated by a JavaBean method `getBMI()` bound to the Web interface using a JSP expression. This is used to create an output SOAP double message.

C. Operation Filtering

After that, the non-pertinent operations of the Web services are eliminated from the starting set according to a collection of filtering constraints. These constraints are boolean expressions which are represented by OCL (Object Constraint Language [14]) expressions that can be added, modified or removed by developers. The OCL expressions are limited to invariants. They navigate in a simple MOF [13] metamodel, which is a simplified excerpt of the UML metamodel (related to operations) extended with some basic features. All expressions have, as a context, an instance of the Operation metaclass. OCL has been chosen because of its simplicity [3] and the existence of a good tool support (OCL Toolkit [4], Eclipse MDT/OCL [6], ...).

An example of a filtering constraint is given below. This OCL constraint states that operations which use the session standard script variable must not be selected.

```
not (self.body.usedType ->includes(t |
  t.name='HTTPSession'))
```

At the end of this step, the developer is asked to choose among the selected operations those which are not pertinent as parts of the published Web services.

D. Operation Distribution

The extracted operations are distributed on multiple Web services based on the following criteria:

1) Spreading Criterion: Similar extracted operations are spread out in different Web services. This ensures some level of reliability and correctness in the published Web services. On the one hand, if a Web service does not provide correct quality requirements (like availability or efficiency) for users of an operation within this service, another service containing a similar operation can be found (reliability). On the other hand, a service should not conceptually (from a correctness perspective in the description of a Web service) provide two or more similar operations (operations that provide the same functionality).

Similar operations are identified following some similarity measures. In the current implementation, a solution based

on operation signatures has been defined. It compares the operations' names, returned type and parameters.

2) Grouping Criterion: In this step, the highly coupled operations are grouped together in a single Web service. This grouping increases at the same time the performance and helps developers in the evolution and maintenance of the generated services. In order to group operations into a single Web service, we measure the coupling between operations by analyzing the static invocations between operations.

E. Composite Web Service Creation

In this step, the potential dependencies between the different selected operations in the Web services are identified. There are two kinds of dependencies between operations: operation invocation dependencies and Web navigation relationships. The first kind of dependencies gives rise to Web service choreographies and the second to Web service orchestrations. These are detailed below:

1) Web Service Choreography Creation.: All calls between operations in the code are captured. If the called operations are published in the same Web service of the caller operation, nothing is done, the calls are left as method invocations. If the called operations are present in the other published Web services, we check if these services are deployed remotely or in the same server, and if they run in the same virtual machine, as the caller service. If the services run in the same execution context, method invocations are left as they are. Otherwise, these operation dependencies are replaced by Web service requests.

In this way, we build composite Web services as code-level choreographies.

In the example presented in Section II, the operation `calculateBMI()` calls some other operations to make arithmetic calculations. These operations (`power(n,m)`, `division(x,y)`, mass and height conversion) are extracted and assembled in two different Web services: `calculationWS` and `conversionWS`. The `calculateBMI()` operation starts first by invoking the mass conversion operation of the `conversionWS` to transform grams into kilograms. This is done through the call to the division operation of `calculationWS`. Then the centimeters are transformed into meters using the height conversion operation (based on the same calculation operation). At last the operations `power(...)` and `division(...)` are called to get the BMI's value. This is a simple but illustrative example of a choreography created starting from the `BMI_Calculator` Web component.

2) Web Service Orchestration Creation.: Navigation documents such as JSF `faces-config` files and their navigation rules are parsed. This allows the identification of the different relationships between Web pages, and potential collaborations of the different Web services extracted from these pages. This task is implemented according to the following algorithm:

```
(01) algorithm WebNavigationParsing {
(02)  let process := ProcessFactory.newInstance();
(03)  for-each(navigRule in WebNavigDocument){
(04)   let opFrom := parseSourceView(navigRule
            .sourceView);
(05)  if(opFrom isNotPreviouslyInvokedInProcess
                process) {
(06)  let op1 :=process.createInvocationTo(opFrom);
(07)   op1.setParameters(variables of process);
(08)   let returnedVal1 := op1.invoke();
(09)   process.store(opFrom,returnedVal1);
(10)  }
(11)  else {
(12)    let returnedVal1 := getStoredReturnedValBy(
             opFrom);
(13)  }
(14)    let exp := navigRule.executionCondition
            .expression;
(15)    let op := process.createInvocationTo(exp);
(16)    let returnedValue := op.invoke();
(17)  if(returnedValue = navigRule
            .executionCondition.value) {
(18)  let var :=process.createVariable(returnedVal1);
(19)  let opTo :=parseDestinationView(navigRule
            .destinationView);
(20)  let op2 := process.createInvocationTo(opTo);
(21)  op2.setParameters(variables in process +var);
(22)  let returnedVal2 := op2.invoke();
(23)  process.store(opTo,returnedVal2);
(24)  }
(25) }
(26)}
```

In this algorithm, we first create a process, and for each navigation rule in the Web navigation document of the parsed Web component, we identify the source operation (Line (04)). The source operation corresponds to the operation that has been generated starting from the navigation's source Web page. The same thing is done for the destination Web page (Line (19)).

As specified in the algorithm, a navigation rule contains three elements: i) a source view (Line (04)), which represents the Web page(s) from which the navigation started (e.g., the page presenting the form for calculating the BMI in the example introduced previously: `bmi.jsp`); ii) a destination view (Line (19)) that corresponds to the Web page(s) to which the user will be automatically directed (e.g., the Web page presenting the interface of the Web component `DietaryAdvisor`: `diet.jsp`); and iii) an execution condition (Lines (14) and (17)) which contains an expression and a value (e.g., the expression is the call to the JavaBean operation for getting the BMI category in the Web component: `#{diet.getBMI_Category}`, and the value is "Obesity").

For each navigation rule, we first test if the source operation has already been called in the process while parsing another navigation rule (Line (05)). This ensures that operation invocations are not duplicated. In the case of an operation which has already been invoked in the process, we just get the returned value (Line (12)). This kind of values are stored after each operation invocation (see Lines (09) and (23)). Then, we compare the value obtained after invoking

the operation defined in the expression of the condition and the value of this condition (lines (16) and (17)). If the two values are equal, then we invoke the corresponding destination operation (Lines (19) to (22)).

In the example presented in Section II, we generate a BPEL (Business Process Execution Language [12]) process (`HealthyDietProcess`). In its description, the process first invokes the `calculateBMI` operation of the first service. Then, it stores the result into a variable and invokes the second operation of the same service to get the BMI category, using the content of the variable as a message. If the returned value, stored in a second variable, is different from "Normal Weight" (is equal to the other three possible BMI categories), the third operation is invoked, using the stored BMI category and other information (dietary habits and eating disorders), to get the corresponding dietary guideline. At last, it invokes the `sendMail` operation with the necessary data.

In orchestration creation, before each operation invocation (see Line (21) in the algorithm above), we prepare the list of parameters. A matching of the variables' names in the orchestration and the arguments of the operation to be invoked is performed. This ensures that arguments are passed in the correct order.

At last, the two Web services, together with the BPEL process, which is exported as a Web service, are deployed in a server and indexed in the Seekda Web service search engine[4].

F. Web Service Deployment and Indexing

All the generated Web services are made available for other developers on the Internet. The validated set of Web services (composite and primitive ones) are deployed on an application server chosen by the developer/administrator of the Web component-based application. All system configuration parameters should be specified in order to perform this activity. This can be fully supported by the prototype tool we developed. The next task in this step is an assistance activity where the developer is asked to log in or sign up for a new account in the Seekda search engine. The services are then indexed in Seekda Web server.

IV. Tool-support for the Proposed Approach

We implemented the proposed solution as a prototype tool called WSGen: Web Service Generator. The components parsed by WSGen are Java Enterprise archives: EARs (Java Enterprise Archives), JARs (Java Archives) and WARs (Web Archives). JSPs, Servlets, JavaBeans, Enterprise JavaBeans and traditional Java classes in these archives are extracted. These files are analyzed to identify the different methods which will represent the potential future operations of the generated Web services. A particular processing is performed on JSPs and servlets to generate new operations. Starting from `request.getParameter(...)` statements, input parameters are generated, and starting from `out.print(...)` and JSP scriptlet expressions, output parameters are created. All tasks related to the parsing and code generation have been implemented upon the `JaxMe Java Source`[5], `HTML Parser`[6] and `JspC`[7] tools. Besides, `MDT OCL` has been used for OCL interpretation. OCL constraints are checked on an Ecore instance (representing the metamodel defined for operations) created starting from the parsed operations' code.

Finally WGen uses a Tomcat server associated with the Apache implementation support for Web services, Axis. Using the generated deployment files WSGen deploys the desired Web services.

V. WSGen By Example

This section illustrates the application of the approach on a larger example. We explain how to create a Web service-oriented solution starting from a simulated version of a real-world Web application. The later represents the Web service search engine Seekda, which indexes a large set of public Web services in the Internet. In its simulated version, this application is considered as a set of interacting Web components. Each one gives a different view on the application.

- The `login` component: allows client's authentication. It asks a user via a form to enter an email and a password.
- The `new_account` component: allows a new user to register in the application. The user is asked to enter an email, a password and re-type the password for validation.
- The `password_recovery` component: asks the user to enter the email address that she/he used to register in Seekda, and a new password. This component sends then an email that allows the user to activate the new password.
- The `web_services_search` component: asks the user to enter keywords for searching Web services. This component provides as a result a list of Web services. Each service is described by the following information: country name, provider name, WSDL URL, WSDL Cache, monitoring date, server name, availability, documentation, tags and user rating.
- The `advanced_web_services_search` component: allows the user to enter search keywords and other search criteria such as: country name, provider name, some specific tags, the number and order of results.

[4]Seekda Website: http://webservices.seekda.com/

[5]Apache Website: http://ws.apache.org/jaxme/js/index.html

[6]HTML Parser at SourceForge: http://htmlparser.sourceforge.net/

[7]Apache Website: http://tomcat.apache.org/download-70.cgi

A. Generated Primitive Web services

The transformation of this Web component generates the following set of primitive Web services [8]:

- `AccountService`: a Web service composed of the following operations:
  - The `_service_login` operation: receives as input two messages of type String (the email and the password of the user). This operation performs the authentification action.
  - The `_service_new_account` operation: receives as input three messages of type String (the email, the password and the repeated password). This operation performs the creation of new client's account.
  - The `_service_password_recovery` operation: has three messages of type String (the email, the new password, the repeated password). It returns a message that indicates to the client that she/he will receive an email containing a link to activate the new password.
- `SearchingService`: a Web service composed of the following operations:
  - The `_service_BasicSearch` operation: receives a message of type String (search keyword). This operation is used to search web services.
  - The `_service_AdvancedSearch` operation: is used for an advanced search. It receives as input a message of complex type, the elements of this complex type are of type String (search keyword, country name, provider name, some specific tags).
  - The `_searchResult` operation: returns to the client a message of complex type composed of elements of String type. These elements represent the information about the searched Web service (country name, provider name, etc.).

B. Generated Composite Web service

Figure 2 depicts the navigation rules between the different views of the Seekda Web application. These relations between Web views are converted to orchestrations of the previous Web services. First, all navigation paths are calculated from the web navigation document. As illustrated in Figure 2 there are sixteen[9] navigation paths for the seekda web application. For each navigation path, a BPEL process is generated using the previous algorithm. Figure 3 shows an excerpt of a generated BPEL process from the navigation path : WelcomeSeekda → new_account → advancedSearch → searchResult. This process represents an orchestration of the corresponding generated services. The interface of the new BPEL composite Web service uses

[8]This is not an exhaustive list. Some other generated operations will be introduced in the following subsection.

[9]Some of them are duplicated and have different sources.

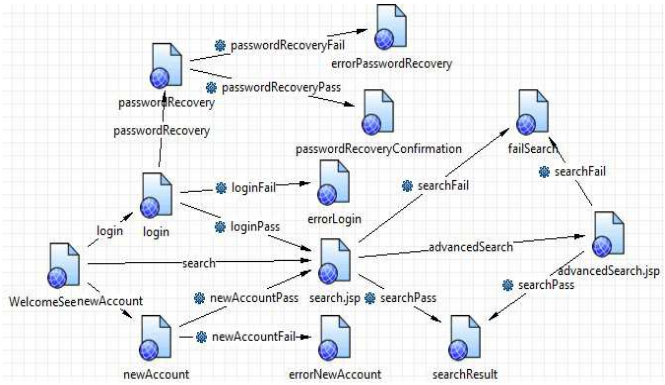

Figure 2. Navigation Rules in the simulated Seekda Web Application

a set of port types, through which it provides operations to clients. The tool generates first, the WSDL file for this interface and their port types. As depicted in Figure 3, the partner link at the left side represents a client of the service provided by the BPEL process. The partner links at the right side represent the Web services (`AccountService` and `SearchingService`) that participate in the BPEL process. We start the process with a "`receive`" activity to receive requests from external clients. These clients represent others applications or Web services that consume the service provided by the BPEL process. We have then an "`invoke`" activity to call the `_service_new_account` operation of the `AccountService`. After invoking this operation, we have in the process another "`invoke`" activity to the`newAccountAction` operation, which performs the creation of a new account in Seekda web application. This returns one of two values `"newAccountPass"` or `"newAccountFail"`, which represent respectively, success or failure of the registration. After that, we have an "`if`" activity, in which, we test whether the returned value is equal to the value `newAccountPass` or not. This value as depicted in Figure 2 represents the value of the condition in the navigation rule. Based on this test, the process invokes the `_service_AdvancedSearch` operation or not. Before invoking this operation,there is a "`receive`" activity which gets from the client the search keywords and other search criteria such as: country and provider names. The next activity in this process is an invoke to the `advancedSearchAction` operation. We have then, an "`if`" activity in which, we compare the returned value of the operation with `"searchPass"` value. If they are equal, the process invokes the `_searchResult` operation. Finally, the returned message from this operation is sent to the client using the "`reply`" activity.

In this way, developers can directly use the services generated from the Seekda search engine application, either for creating new accounts or for searching Web services.

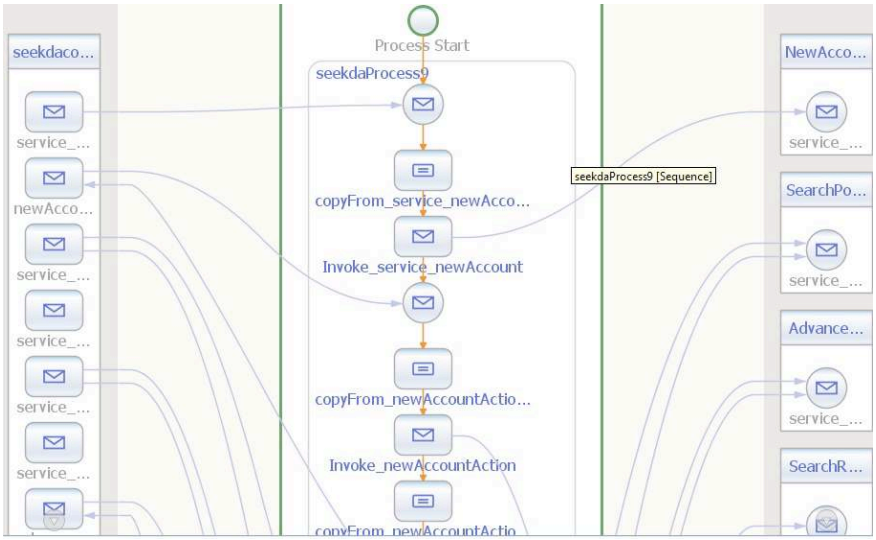

Figure 3. Excerpt of the generated BPEL Process

They can build extensions of these services to provide more sophisticated solutions. For example, in the tools implemented by our team [1], we classify hierarchically the result set of Web services obtained from Seekda, in order to make search and browsing easier. The set of Web services returned by the Seekda application consists of HTML pages. Instead of building an HTML parser "from scratch" to analyze each HTML page, we can consider here these tools as extensions to the functionalities provided by the Web interfaces of Seekda application accessed through our generated Web services. In this way, these tools can simply send requests to the Web service `SearchingService` and based on the obtained result, they classify Web services.

VI. Related Work

In [10], Roger Lee et al. developed an approach for converting functionalities implemented in software components into Web services. [11] adopt an extensible mechanism to transform components implemented in different programming languages. These two works allow to a client to specify a request for searching a given functionality in components developed with different programming languages (C++ or Java) and deployed in a Web server or located in repository. As an answer to this request, services are generated automatically for the desired functionalities. Compared to these reactive systems, WSGen is proactive. It does not react to a client request, but it allows a web application engineer during the deployment of her/his component-based system to anticipate the export of some functionalities to third party developers as Web services. In addition, in WSGen we deal with Web interface conversion into stateless Web services. However in [10] and [11], only business functionalities implemented in software components are transformed.

Wike [8] generates virtual Web services by the extraction of information from Web pages. Users can define patterns which are used to extract partial information from Web pages. The extraction function can be used to generate a Web service that returns the result of the extraction process. Content-based Web pages are not the main concern in our approach. Indeed in our process, Web components including Web interfaces and business logic implementation are the artifacts concerned by Web service generation. Wike is however a complementary solution to our work. Web services that are generated using our approach starting from Web components, which produce to users during execution a large quantity of content, can be enhanced with new operations that return only partial information (texts, images, ...) using Wike. Invocations to these new operations can be added to the orchestrations generated by WSGen.

Many works in the literature proposed some model-driven techniques to generate Web service-oriented applications. Bauer and Müller [2] applied MDA[10] to provide a mapping of elements from UML2 sequence diagrams (considered as PlMs – Platform Independent Models) to a representation of compositions of Web services using BPEL (considered as PSMs – Platform Specific Models). This approach aims in particular at reducing the complexity of building BPEL specifications manually. [7] proposed a model-driven process for building Web service compositions. This process transforms WSDL descriptions into UML models. These models are integrated by the developer to form composite Web services, which contain interface and workflow descriptions. Interface models are described using stereotyped UML class diagrams and workflow models are represented by stereotyped activity diagrams. The resulting composite services are exported at

[10]OMG's Website: http://www.omg.org/mda/specs.htm

last as WSDL descriptions. This work provides means for making forward engineering (UML to WSDL and BPEL) and reverse engineering (WSDL to UML) by specifying bidirectional transformation rules. Another model-driven approach for creating service-oriented solutions has been proposed in [9]. In this work, a UML profile has been defined for service-oriented applications.

All these works are complementary to our approach. In our work the transformations are made from PSM to PSM. Web components, which are models specific to a given platform (in the current implementation, Java EE), are converted into Web services, which are considered as another platform-specific model (WSDL, Java and BPEL, in the actual version of WSGen). The UML profile presented in [9] can be used to define high-level models of the generated Web services. The other approaches can be used to make a reverse engineering of the generated Web services or orchestrations and obtain more understandable models (compared to code). In addition, all these related works focus on UML modeling and generating new Web services starting from models of a high level of abstraction. In our approach, we worked on the transformation of existing Web code.

VII. Conclusion and Future Work

Recent research in software and information systems engineering emphasizes the need for the proposition of new languages, methods, and tools for building systems by shifting from a product-centric to a service-oriented view [5]. In this paper, we present a process for the transformation of Web component-based applications into Web service-oriented ones. Web components are seen here as software artifacts embedding business logic code and exporting Web interfaces. This kind of modules are analyzed, the different elements that compose them are extracted to identify different Web services and possible collaborations of these services. All of the resulting services are deployed on the Internet in order to allow third party development. In our work, we consider these deployed artifacts (embedding Web interfaces) as remote APIs (as services) that offer the opportunity for developers to extend the functionality provided by these services and exploit the resources used by them.

Some enhancements will be made on the transformation process implemented in WSGen. We are now working on the implementation of more sophisticated techniques for grouping complementary operations in Web services, based on “text-mining” of Web components’ documentation. At the conceptual level, we plan to study the formalization of the performed transformation as a set of high level declarative rules. We then define such rules in a MOF/QVT-compliant language [15] and thus integrate our solution in a Model-Driven Engineering process. At the tool level, we plan to work on the generation of high-level specifications of choreographies in ”WS-CDL”-compliant languages starting from collaborations of the generated Web services’ operations. Experimenting in depth our approach by measuring its scalability and performance on large Web applications is another perspective of our work.

References

[1] Z. Azmeh, M. Driss, F. Hamoui, M. Huchard, N. Moha and C. Tibermacine. Selection of Composable Web Services Driven by User Requirements. In *Proc. of ICWS*, 2011.

[2] B. Bauer and J. P. Müller. Mda applied: From sequence diagrams to web service choreography. In *Proc. of ICWE’04*, pages 132–136. Springer-Verlag, 2004.

[3] L. C. Briand, Y. Labiche, M. Di Penta, and H. D. Yan-Bondoc. An experimental investigation of formality in uml-based development. *IEEE TSE*, 31(10), 2005.

[4] T. U. Dresden. Ocl compiler web site. http://dresden-ocl.sourceforge.net/, 2009.

[5] A. Finkelstein and J. Kramer. Software engineering: a roadmap. In *Proc. of ICSE’00*, pages 3–22. ACM, 2000.

[6] Eclipse Foundation. Model development tools website. http://www.eclipse.org/modeling/mdt/?project=ocl, 2009.

[7] R. Gronmo, D. Skogan, I. Solheim, and J. Oldevik. Model-driven web services development. *International Jounal of Web services Research*, 1(4), 2004.

[8] H. Han and T. Tokuda. Wike: A web information/knowledge extraction system for web service generation. In *Proc. of ICWE’08*, pages 354–357. IEEE CS, 2008.

[9] S. K. Johnson and A. W. Brown. A model-driven development approach to creating service-oriented solutions. In *Proc. of ICSOC’06*, pages 624–636. Springer, 2006.

[10] R. Lee, A. Harikumar, C.-C. Chiang, H.-S. Yang, H.-K. Kim, and B. Kang. A framework for dynamically converting components to web services. In *Proc. of SERA’05*, 2005.

[11] A. Marinho, L. Murta , and C. Werner Extending a Software Component Repository to Provide Services. In *Proc. of ICSR’09.* Falls Church, USA, pp. 258-268, 2009.

[12] OASIS. Oasis consortium website. Web Services BPEL Version 2.0: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, 2007.

[13] OMG. Meta object facility (mof) 2.0 core specification, document ptc/04-10-15. OMG Website: http://www.omg.org/cgi-bin/apps/doc?ptc/04-10-15.pdf, 2004.

[14] OMG. Object constraint language specification, version 2.0, document formal/2006-05-01. OMG Website: http://www.omg.org/cgi-bin/apps/doc?formal/06-05-01.pdf

[15] OMG. Meta object facility (mof) 2.0 query/view/ transformation specification, version 1.0, document formal/2008-04-03. OMG Website: http://www.omg.org/spec/QVT/1.0/PDF/

[16] W3C. Web services choreography description language version 1.0, w3c candidate recommendation. W3C Website: http://www.w3.org/TR/ws-cdl-10/, 2005.