# From Web Components to Web Services: Opening Development for Third Parties

Chouki Tibermacine[1] and Mohamed Lamine Kerdoudi[2]

[1] LIRMM, CNRS and Montpellier University, France
[2] Computer Science Department, University of Biskra, Algeria
Chouki.Tibermacine@lirmm.fr , l.kerdoudi@univ-biskra.dz

**Abstract.** One of the main advantages of the Web component-based development 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. Besides this, since many years, Web services confirmed their status of the most pertinent solution for a given service provider, like Google, Amazon or FedEx, to open its solutions for third party development. In this paper, we present an approach to build web services starting from existing web component-based applications 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 EE technologies.

## 1 Introduction: Context and Motivation

Web component-based development aims at decoupling Web application code modules, and making them reusable and customizable software entities. Indeed, in this paradigm a step has been taken forward in modularizing Web applications and thus separating business logic code, from view, model and controller one. One of the technologies leading this field is Java EE[3] and its numerous frameworks like Struts or JSF. Web components in such technologies are entities that can be used and reused in different applications and customized according to the application requirements.

In our opinion, these technologies are currently the ideal solutions for developing large and complex applications with highly critical requirements on maintainability. 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. In this paper, we present an approach (see Section 2) 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 offer the opportunity to other developers to build extensions of the services provided by their artifacts.

[3] Java Enterprise Edition from Sun Microsystems: http://java.sun.com/javaee/

This transformation goes through a process composed of several steps, which has been implemented 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 compositions of these services are provided at output. The paper ends by discussing some related work and presenting our future work.

## 2 Proposed Approach

The transformation process is composed of five steps:
**1. 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 server-side programs (JSP pages, for example) is grouped within a single operation and formatted to be executed as stand-alone code. Similar extracted operations are spread out in different Web services using a simple similarity measure (based on operations' names, parameters, ...).
**2. Input and Output Message Identification:** The input and output messages related to each operation in the Web services are extracted starting from the parsed elements in the Web component. For classes and other structured code elements, the parameters and the returned value are formatted as SOAP messages. The code present in other programs (like JSP pages) is parsed to extract the input values received in the HTTP requests. Their types are inferred from the parsed code by analyzing type casts and other conversions. The contents produced by theses programs, which are viewed at the client side (like JSP expressions), are considered as returned values.
**3. Operation Filtering:** After that, the non-pertinent operations of the Web services are eliminated from the starting set, according to a collection of filtering heuristics. These heuristics are boolean expressions which are represented by OCL expressions that can be added/modified/removed by developers. An example of a filtering heuristic is: "Operations that use the session standard script variable are not taken into account". This is formalized in OCL as follows:
`not (self.body.usedType->includes(t|t.name='HTTPSession'))`
At the end of this step, the developer is asked to choose the less pertinent operations to remove from the Web services that will be published.
**4. 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-call dependencies (which give rise to Web service choreographies) and Web navigation relationships (which allow to generate Web service orchestrations).
**4.1. Web Service Choreography Creation:** All calls to operations in the code are captured. If the called operations are published in the same web service of the caller operation, the calls are left as method invocations. If they are published on a remote host, these operation dependencies are replaced by Web service requests. In this way, we build composite Web services as code-level choreographies. In the near future, we plan to generate high-level choreographies in "WS-CDL"-compliant languages [7].

**4.2. Web Service Orchestration Creation:** Navigation documents such as JSF `faces-config` files 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 a simple algorithm in which we first create a process, and for each navigation rule in the Web navigation document, we identify the source and destination operations. Before each operation invocation, we prepare the list of parameters.
**5. Web Service Deployment and Indexing** 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. If the developer deploys the Web services in different servers, the dependencies between collaborative operations in composite services are resolved. Then, on the Seekda Web server, we create an account and register the services. Once the services indexed, we propose to the developer a smart mechanism of keyword extraction, based on our previous work [2]. This mechanism identifies potential tags for the services starting from their WSDL description. The keyword list (based on identifiers in the service operation names, parameters' names, ...) is proposed to the developer in order to select tags for the deployed services.

## 3 Related Work

In [5], Roger Lee et al. proposed an approach which allows a client to specify a request for searching a given functionality in components deployed in a Web server. As an answer, a Web service or a composition of Web services is generated automatically. Our approach is proactive; it does not react to a client request, but it allows a web application engineer to anticipate the export of some functionalities to third party developers as Web services. In addition, in our work we deal with Web interface conversion into stateless Web services. In Wike [3], developers can define patterns for extracting partial information from Web pages. These patterns are encapsulated in functions that can be exported as Web services. In our work, content-based Web pages are not the main concern. 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.

Many works in the literature proposed model-driven techniques to generate Web service-oriented applications. Bauer and Müller [1] developed an approach to map 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). In [8], the authors propose transformation rules for converting orchestration models specified in CCA (Component Collaboration Architecture), which is part of the UML profile for Enterprise Distributed Object Computing (EDOC [6]), into BPEL specifications. Another model-driven approach for creating service-oriented solutions has been proposed in [4]. In this work, a UML profile has been defined for the expression of service-oriented applications. 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). The UML profile presented in [4] 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 the code).

## 4 Conclusion: Contribution and Future Work

We implemented the proposed solution as a prototype tool called WSGen: Web Service Generator. The components parsed by WSGen are Java Enterprise archives. JSPs, Servlets, JavaBeans, Enterprise JavaBeans and traditional Java classes in these archives are extracted. To deploy services, this tool uses a Tomcat/Axis server.

In our work, we consider the generated and deployed Web services as remote APIs that offer the opportunity for developers to extend the functionalities provided by these services and exploit the resources used by them. Many enhancements still have to be performed in the transformation process implemented in WSGen. We are now working on the implementation of more interesting techniques for grouping complementary operations in Web services, based on "text-mining" of the Web components' documentation. At the conceptual level, we plan to study the formalization of the process as a set of high level declarative (or a combination of declarative and imperative) transformation rules, and thus integrate our solution in the Model-Driven Engineering world. At the tool level, we plan, as introduced previously, to work on the generation of high-level specifications of choreographies in "WS-CDL"-compliant languages [7] starting from collaborations of the generated Web services' operations.

## References


1. 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.
2. J.-R. Falleri, Z. Azmeh, M. Huchard, and C. Tibermacine. Automatic tag identification in web service descriptions. In *Proc. of WEBIST'10*, April 2010.
3. 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.
4. 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.
5. R. Lee et al. A framework for dynamically converting components to web services. In *Proc. of SERA'05*, 2005.
6. OMG. Uml profile for enterprise distributed object computing (edoc). OMG Website: http://www.omg.org/technology/documents/formal/edoc.htm, 2004.
7. W3C. Web services choreography description language version 1.0, w3c candidate recommendation. W3C Website: http://www.w3.org/TR/ws-cdl-10/, 2005.
8. X. Yu, Y. Zhang, T. Zhang, L. Wang, J. Zhao, G. Zheng, and X. Li. Towards a model driven approach to automatic bpel generation. In *Proc. of ECMDA*, pages 204–218. Springer-Verlag, 2007.