MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ MOHAMED KHIDER - BISKRA

N° d'ordre: .....
Série: .....

— **FACULTÉ DES SCIENCES EXACTES ET DES SCIENCES DE LA NATURE ET DE LA VIE** —

— **DÉPARTEMENT D'INFORMATIQUE** —

THÈSE

présentée pour obtenir le diplôme de

DOCTORAT EN SCIENCES

SPÉCIALITÉ : **INFORMATIQUE**

From Web components to Web services: opening development for third parties

Des composants Web aux Web services : ouvrir le développement aux tierces parties

par

Mohamed Lamine KERDOUDI

Soutenue le 26/05/2016, devant le jury composé de :

Abdelmalik BACHIR, Professeur, Université de Biskra, Algérie ..... Président
Salah SADOU, Maître de Conférences HDR, IRISA, Université Bretagne Sud, France ..... Rapporteur
Chouki TIBERMACINE, Maître de Conférences, LIRMM, Université Montpellier II, France ... Co-Rapporteur
Abdelkrim AMIRAT, Professeur, Université de Souk Ahras,Algérie, ... Examinateur
Foudil CHERIF, Professeur, Université de Biskra, Algérie ..... Examinateur
Salim BITAM, Maître de conférences A, Université de Biskra, Algérie ..... Examinateur

Contents

Acknowledgement

Saying thank you is more than good manners. It is good spirituality.

Alfred PAINTER.

First and foremost, praises and thanks to my God, Allah for everything.

I would like to express my sincere gratitude to my supervisor Assoc. Prof. Dr. Salah Sadou for accepting the supervision of this thesis. Thank you for all the guidance, patience, immense knowledge, and the support that you gave me at every stage in my Ph.D. Project.

I express my profound gratitude to my co-supervisor Assoc. Prof. Dr. Chouki Tibermacine for their continuous support of my Ph.D study and related research, for their patience, immense knowledge. Their guidance helped me in all the time. You have supported me at every stage in this thesis. Thank You!

My sincere thanks also to the committee members: Prof. Abdelmalik BACHIR, Prof. Abdelkrim AMIRAT, Prof. Foudil CHERIF, and Dr. Salim BITAM, for accepting to evaluate this work. I would like to take this opportunity to express my deepest gratitude to Prof. Brahim Mezerdi for all the help and the support that he gave me in all the time. Thank you!

I sincerely thank all the members of LIRMM laboratory especially the MAREL team for the numerous helpful discussions during my stay in Montpellier. I also thank my colleagues at the computer science department of Biskra university. Many thanks to my friends for all the help and the support they gave me in every stage of my research study.

Words cannot express how grateful I am to my father and my mother for all the sacrifices that they made for me. They have been with me throughout my life and whose love brought me where I am today. I owe a lot to both of you. I am very grateful to my brothers, my sisters, and all my family members for all the support and the help they provided throughout my life. Thank you very much.

Abstract

Web applications are nowadays prevalent software systems in our everyday's life. A lot of these applications are built mainly by assembling Web components. These components are first customized to meet the requirements of the built applications, then instantiated and assembled with other component instances. The Web applications are then deployed in a Web server in order to be tested and validated. Finally, they are put into production by deploying them in a Web/application server to make them accessible for end users only. Thus, they are not designed by considering future extensions that would be developed by third parties. One possible and interesting solution for opening these applications for such kind of extension development is to create and deploy Web services starting from these applications. This thesis addresses the problem of opening Web Applications for third party development. We proposed a set of methods and tools that contribute in the migration of Web Component based applications toward Web service oriented systems. Firstly, we proposed a formal model that represents in unambiguous way the source systems (Web applications) and the target systems (service-oriented systems). This formal definition helps in understanding both kinds of systems and it enabled us to present more accurately the migration approach. Secondly, we proposed a method to generate operations that are published in Web services for each functionality provided by a Web application. In addition, it generates new operations starting from Web interfaces. Thirdly, we developed another complementary method to generate executable orchestrations, as BPEL (Business Process Execution Language) processes, starting from navigations in the Web interfaces of these applications and to create Web service choreographies starting from the dependencies between Web components. Fourthly, we proposed an approach for recovering high level specifications in BPMN(Business Process Model and Notation) and in SCA (Service Component Architecture) starting from the collaborations between the generated Web services. These architectures help in better understanding the service compositions. Finally, in order to evaluate the performance and the accuracy of the proposed approaches, we implemented and experimented the solution in the migration of three real-world Web applications toward Web service-oriented systems.

Keywords: Component-Based Web Applications, Service Oriented Architecture, Web Service, Service Composition, SOA Migration, Software Architecture Recovery, Reverse Engineering.

Résumé

Aujourd'hui, les applications Web sont des systèmes logiciels qui dominent notre vie quotidienne. Un grand nombre de ces applications sont construites par l'assemblage de composants Web. Ces composants sont d'abord personnalisés pour répondre aux besoins des applications développées, puis instanciés et assemblés avec d'autres instances de composants. Les applications Web sont ensuite déployées dans un serveur Web afin d'être testées et validées. Enfin, elles sont mises en production en les déployant dans un serveur Web /d'application pour les rendre accessible seulement par les utilisateurs finaux. Donc, elles ne sont pas conçus en prenant en considération des futures extensions qui pourraient être développées par des tierces parties. Une solution possible et intéressante pour ouvrir ces applications pour ce type de développement d'extensions est de créer et de déployer des services Web à partir de ces applications. Cette thèse étudie le problème d'ouverture des applications Web pour le développement tiers. Nous avons proposé un ensemble de méthodes et d'outils qui contribuent à la migration des applications à base de components Web vers des systèmes orientés services Web. Tout d'abord, nous avons proposé un modèle formel qui représente clairement les systèmes sources (applications Web) et les systèmes cibles (systèmes orientés services). Cette définition formelle aide à comprendre les deux types de systèmes et elle nous a permis de présenter plus précisément l'approche de migration. Deuxièmement, nous avons proposé une méthode pour générer des opérations qui sont publiées dans des Web services pour chaque fonctionnalité fournie par l'application Web. En outre, elle génère des nouvelles opérations à partir des interfaces Web. Troisièmement, nous avons développé une autre méthode complémentaire pour générer des orchestrations exécutables, comme des processus BPEL, à partir des navigations dans les interfaces Web de ces applications et de créer des chorégraphies de services Web à partir de dépendances entre les composants Web. Quatrièmement, nous avons proposé une approche pour récupérer des spécifications de haut niveau en BPMN et en SCA à partir des collaborations entre les services Web générés. Ces architectures aident à mieux comprendre les compositions de services. Afin d'évaluer la performance et la précision des approches proposées, nous avons implémenté et expérimenté la solution dans la migration de trois applications Web du monde réel vers des systèmes orientés services Web.

Mots clés: Applications à base de Composants Web, SOA, Services Web, Composition de Ser-

vices, Migration vers SOA, Extraction d'Architecture Logiciel, Rétro-ingénierie.

ملخص

تمثل تطبيقات الويب في الوقت الحاضر البرمجيات والنظم السائدة في حياتنا اليومية؛ حيث أن هناك الكثير من هذه التطبيقات مبنية أساسا عن طريق تجميع مكونات الويب ، حيث يتم أولا تخصيص هذه المكونات لتلبية متطلبات التطبيقات التي تم بناؤها، ثم يتم تجميعها مع مكونات الويب الأخرى، بعدها يتم نشر تطبيقات الويب في خادم ويب من أجل فحصها والتحقق من صحتها، وأخيرا توضع قيد الإنتاج من خلال نشرها في خادم ويب أو خادم تطبيق؛ لجعلها في متناول المستخدمين النهائيين فقط، وبالتالي لا يتم الأخذ بعين الاعتبار عند تصميمها التمديدات المستقبلية؛ التي قد يتم تطويرها من قبل أطراف ثالثة، وأحد الحلول الممكنة والمثيرة للاهتمام لفتح هذه التطبيقات لمثل هذا النوع من تطوير التمديدات هو إنشاء ونشر خدمات ويب بدءا من هذه التطبيقات. وتتناول هذه الأطروحة دراسة مشكلة فتح تطبيقات الويب من أجل تمكين أطراف ثالثة من تطوير تمديدات، حيث تم اقتراح مجموعة من البرمجيات، وكذا مناهج جديدة تساهم في هجرة التطبيقات القائمة على مكونات الويب نحو التطبيقات القائمة على خدمات الويب. أولا، اقترحنا نموذج رسمي (شكلي) الذي يمثل بطريقة غير مبهمة أنظمة المصدر (تطبيقات الويب) والأنظمة المستهدفة (خدمات ويب). هذا النموذج الشكلي يساعد في فهم كلا النوعين من الأنظمة و قد مكننا من شرح بأكثر دقة نهج الهجرة المقترح. ثانيا، اقترحنا منهج يقوم بإنتاج مجموعة من العمليات التي يتم نشرها في خدمات ويب لكل وظيفة مقدمة من طرف تطبيق الويب، وبالإضافة إلى ذلك فإنه ينتج عمليات بدءا من واجهات الويب. ثالثا، قمنا بتطوير منهج آخر مكمل لإنتاج توزيعات أوركسترالية لخدمات الويب قابلة للتنفيذ؛ وذلك بدءا من التصفح في واجهات الويب الموجودة في هذه التطبيقات، ولإنشاء توزيعات كريوقرافية لخدمات الويب وذلك بدءا من العلاقات الموجودة بين مكونات الويب. رابعا، اقترحنا نهجا لاستخراج نماذج هندسية عالية المستوى مبنية من التعاون الموجود بين خدمات الويب التي تم إنشاؤها. هذه النماذج الهندسية تساعد في فهم أفضل لخدمات الويب المدمجة. وأخيرا ومن أجل تقييم الأداء ودقة المناهج المقترحة، لقد نفذنا وقيمنا الحلول المقترحة في هذه الأطروحة من خلال القيام بتحويل ثلاثة تطبيقات ويب موجودة في العالم الحقيقي نحو أنظمة خدمات الويب.

الكلمات المفتاحية: : التطبيقات بمكونات الويب، البنية الخدمية ، خدمة ويب، دمج الخدمات، الهجرة الى البنية الخدمية، استخراج هندسة برمجيات ، الهندسة العكسية.

CHAPTER 1

Introduction

To program is to understand.

Kristen NYGAARD.

Preamble

This introduction presents the context of our work, the problem studied, and the contributions of this dissertation. The context concerns the Web component-based development, the software reuse and component-based development and finally the service oriented development. The problem studied in the dissertation and our contributions are presented through a set of research questions and a series of challenges that we aim to achieve. Finally, we end this introduction by stating the organization of this dissertation and the list of our publications that are related to this work.

1.1 Context

WEB APPLICATIONS are software systems that are widely used since the early nineties and the emergence of the World Wide Web. They have gained a lot of popularity comparatively to Desktop applications, because of their ease of use, *via* Web browsers, whereas Desktop applications need sometimes heavy installations. These applications provide to their users Web interfaces through which they can submit data to the server-side scripts and through which they can receive the processing results.

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. These technologies are currently one of the most interesting solutions for developing large and complex applications with highly critical requirements on maintainability and portability. 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.

Besides, reuse is one of the major objectives of software engineering. Research on software reuse has greatly evolved in recent years, from code reuse to knowledge reuse and software component reuse. Unlike traditional development approaches, the development of a new system is not made from scratch and does not require recreating large parts of the system for each new change. For a decade, there is a growing interest in software engineering for developing techniques and tools to build applications by assembling software components [Szyperski *et al.*, 1999]. This interest in components is resulting from the desire to reduce development costs by increasing the reuse instead of inventing new forms of development to face the increasing complexity of applications related to new requirements such as reliability and evolution. Currently there are many proposed definitions of the concept of software component, the most cited definition is:

"A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties." [Szyperski, 2002]

This definition can be interpreted on different levels of abstraction. Here, we explain some parts of this definition: i) "a unit of composition": This means that components are developed to be composed with other components. ii) "contractually specified interfaces": It provides an

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

interface comparable to a traditional application programming interface (API) through which it exposes public capabilities as methods in order to be invoked by other programs. iii) “can be deployed independently”: A component is self-contained. Changes to the implementation of a component do not require changes (or a re-installation) to other components. iv) “third parties”: Persons who assemble applications from components are not necessarily the same as those who created these components. They can build new applications by extending the functionality provided by the components.

Nowadays, SOA paradigm is considered as one of the best solutions for developing systems by reusing a set of independent and loosely coupled software entities, called “**Services**” [Erl, 2009]. Services are a natural development of software components. They encapsulate discrete functionality, which can be distributed and accessed programmatically. Besides, one possible and usual way of implementing SOA architectures relies on Web services. Web services are functionalities based on standard Internet and XML based languages and protocols 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 way, Web service providers, which hold some precious resources (like large databases of products to retail of Amazon, or weather forecast data of Meteo France), offer third party developers the opportunity (for free or not) to build new applications by extending their public services, and thus capitalize on these resources.

This thesis addresses the problem of Opening Web Applications for Third Party Development that is precisely defined in the next section.

1.2 The problem studied in the thesis

The majority of Web applications have been designed and deployed exclusively for end users who are humans. They have not been considered as a possible basis for remote extensions by third parties. However, after deploying a Web application within an application server, there is no means to directly publish some functionalities of the application for third party development. To extend these applications, third-party developers have no other choice than making HTTP requests in their programs and then parsing the HTML code returned by these Web applications. This represents a cumbersome task for a developer especially that in most cases the parsed HTML code is too long and verbose. In addition, HTTP requests need frequently a detailed customization, and HTTP responses need careful handling (dealing with errors and redirections).

In this thesis, we are interested in studying the following general question:

How can third party developers extend easily and efficiently the business logic implemented by an existing Web component-based application?

In other words, the problem tackled here is to find a way to assist a developer to open its Web application[2] (or some of their functionalities), thereby allowing it to be easily and efficiently i) re-used and extended remotely by third parties for developing new systems; ii) integrated with existing systems ; iii) accessed from any program or device. As stated in the context of this introduction, SOA paradigm is considered as one of the best solutions for opening software systems to be reused and integrated with new or existing systems. As for Web services, they 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. Therefore, we redefine the problem of opening Web applications for third party development, as a problem of searching a way for assisting efficiently the developers to migrate their Web application (or some of their functionalities) toward Web service oriented solutions. We divided and defined the problem of opening Web applications toward Service Oriented Solutions through the following sub-questions:

RQ1: *How to define precisely and in an unambiguous way the used concepts in the source system (Component-based Web application) and the target system (Web service oriented system) ?*

In order to define and present more accurately the processing to be performed on Web applications to generate Web service oriented systems, we need to specify in unambiguous way (mathematically) the used concepts in both systems. Moreover, we need to identify properties that characterize both systems and provide a formal description that represents them.

RQ2: *What are Web application capabilities which can be thought of remote Web services, and how to migrate these capabilities to Web services?*

We need to identify what are Web application functionalities which can be migrated into Web services. In addition, we need to identify what are the activities and their sequencing that the developer has to follow in order to generate pertinent operations and Web services starting from a Web application.

RQ3: *How to compose automatically and in a rational way the generated individual Web services in order to provide coarse grained functionality?*

The functionalities that are migrated into Web services originally collaborate together in the Web application in order to provide coarse grained capabilities to end-users. Unfortunately,

[2] We consider in this thesis existing “white-box" Web applications, whose source code can be migrated.

the third party developer cannot easily and efficiently identify these collaborations (because they are embedded in the source code) in order to provide these coarse grained capabilities as additional Web services. The third party developer has no other choice than parse manually the source code to identify this collaboration and composing these services manually. This tasks are error-prone, cumbersome, and so time-consuming. Indeed, despite the existence of languages and standards for composing services such as BPEL [OASIS., 2007], the Web service composition still is a highly complex task, and it is already beyond the human capability to deal with the whole process manually [Rao et Su, 2005]. Hence, we need to find a way to compose automatically and in rational way the generated Web services.

RQ4: *How can we help third party developers or maintenance engineers to understand and evolve easily the generated service oriented application?*

Knowing that all software is required to regularly change (Lehman's 1st law of evolution [Lehman et Belady, 1985]) and before starting the evolution a good understanding of the implementation is required. The change in these compositions of services is often carried out with an ad-hoc way directly in code. This makes very difficult the understanding of the rationale behind a given composition of services, especially for large applications. In addition, it implies a direct impact on the cost and the risk of errors during an evolution. According to [Bennett, 1996], in the software evolution, the software understanding stage costs more than 50% of the maintenance time. Hence, in order to understand and maintain these large applications, it is helpful to know their architectures. Unfortunately, the created (Web) service oriented composites have not explicit service oriented architectures. In order to improve the understanding for developers, we need to define in this thesis a way to explicit the hidden service oriented architecture from the source code of a (Web) service oriented application. The service architectures represent the systems using service-oriented concepts. They explicitly show to developers the collaboration between (Web) services, the exchanged messages, and the relationships between service providers and service consumers and their related roles.

1.3 Contributions

We proposed a systematic method that assists developers to migrate their Web applications toward Web service oriented systems. In this way, we make the functionalities exposed by Web interfaces of the application accessible as Web services for remote extensions. In order to develop our method, we need to answer the previous asked sub-questions that are related to the SOA migration problem. Each answer to these questions corresponds to one of our contributions in this thesis. We summarize these contributions as follows:

1. **Formal model for Web applications and Service oriented Systems**:

   In the first contribution, we defined a formal model that represents the different concepts used in our approach. Indeed, this formal model describes in an unambiguous way what composes Web applications, which are the input of our method, and the Web service-oriented systems, which are the output. Both systems are represented as two directed graphs expressed using a set-theoretic notation. The properties that characterize the concepts in both systems are mathematically defined as sets. The provided formal descriptions are used to better understand both kinds of software systems. In addition, it enabled us to present more accurately the processing performed on Web applications to generate Web service systems. Therefore, the transformations of Web applications to Web services can be expressed as a mapping between graphs. The mapping is explained using a set of procedures and functions that use the proposed formal model.

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

   The second contribution is the heart of this thesis. We proposed a method to assist systematically the developers so that they can identify and generate easily Web services starting from their existing component-based Web applications. It helps them to generate operations that are published in Web services for each functionality provided by a Web application (methods and functions in the server-side source code of the application). In addition, it generates new operations starting from Web user interfaces. This makes it possible to provide parameterizable services starting from pages designed for human interactions. This transformation goes through a semi-automatic multi-step process. First, a parsing of the different Web component elements is performed to extract the potential set of Web services. Then, the input and output messages related to each Web service are deduced starting from the parsed elements. After that, the non-pertinent operations in the Web services are eliminated from the starting set according to a set of filtering constraints and the intervention of the developer. In the next step, the identified operations are distributed into Web services based on the metrics of coupling and similarity between operations so that we assure a good level of granularity for these services.

3. **Generation of composite Web Services**:

   In the third contribution, we developed another complementary method to generate a set of composition of the generated Web services starting from the dependencies that exist between the Web components (that have been transformed into individual primitive Web services) of a Web application. An automatic parsing of the source code of the Web application is performed to identify these dependencies.

   Actually, there are two kind of dependencies, those which are between Web interfaces and those which are embedded in the source code as method invocations.

The first kind represents the end-users navigation in the Web interfaces of the application. In our method, the relationships between these Web interfaces are transformed into a set of Web service orchestrations. These orchestrations are generated as BPEL [OASIS., 2007] processes. They implement a coarse-grained functionality provided by the Web application, comparatively with the individual Web services that implement fine-grained functionality. The BPEL is an OASIS standard, considered as one of the leading languages used for implementing Web service orchestrations.

The second kind of dependencies represents the collaborations between the generated Web services at code level. In our method, these dependencies are transformed into a set of Web service choreographies. To do so, we parse the source code of the generated Web services in order to identify these dependencies and transform them into Web services requests.

4. **Recovering Architectures from Service Oriented Systems**:

   In the fourth contribution, we proposed a reverse engineering approach for recovering high level specifications of the generated services. Our approach generates behavioral and structural models. In the behavioral models, choreographies are specified in BPMN [OMG., 2011a]. In the structural models, choreographies are specified using the SCA specifications [Beisiegel *et al.*, 2009]. The creation of these service architectures helps in better understanding the composition of services, which can be seen at code level only. The proposed recovering approach is applied on the generated Web services starting from the Web application. The recovered architectures can be provided to third party developers to help them in understanding the target application before developing extensions. In addition, they can serve as documentations for future evolution made by the maintenance engineer.

   We studied the application of our approach on large scale systems, where the number of services is high. We chose the OSGi (Open Services Gateway Initiative) applications [McAffer *et al.*, 2010] as a good example of large Service Oriented Applications. This choice allow us to scale to real-world SOA applications, where the number of services is more than thousands.

5. **Implementation**:

   We implemented our methods on a collection of Java Frameworks. We have developed two prototypes: a tool called WSGen which covers the implementation of Contributions 2 and 3, and a tool called ArchGen implementing Contribution 4.

   We focused in our work on a particular kind of Web applications, which are modern component-based Web applications built with Java EE and its frameworks like JSF. They are the input of WSGen tool and a set of (primitive and composite) Web services are provided as output. The choice of such technologies is motivated by the fact that they offer

a structured organization of the source code of Web applications. This made easy the parsing performed in our approach in order to generate Web services. Besides, (Web) Service-based Applications source code (in the current implementation, Java Web services) are the inputs of ArchGen tool and a set of *BPMN* and *SCA* models are provided as output.

6. **Experimentation**:

   Finally, we conducted two experiments for evaluating our proposals. In the first experimentation, we have evaluated the performance and the accuracy of the proposed methods in the migration of three real-world Web applications towards Web service-oriented systems. In the second experimentation, we measured what is the additional cost induced by the using of our proposed approach. We have made some measures in the development of the same software extensions developed first without our approach, and then with our approach. Then, we have made a comparison between the obtained results.

1.4 Thesis Outline

The rest of this thesis is organized as follows: **Chapter** 2 introduces a brief background about Web component-based development, service oriented computing, and particularly the Web services development and their composition. **Chapter** 3 provides an overview of the state-of-the-art in three domains that are related to our work: (i) migrating existing systems toward service oriented solutions, (ii) Web service composition, and (iii) software architecture recovery. **Chapter** 4 starts by introducing a concrete example of a Web application which serves as a running example for illustrating our proposals throughout this thesis. After that, this chapter describes our proposed formal description of the context of this work, which is composed of Web applications (inputs of our SOA migration approach) and Web service-oriented systems (outputs of our approach). Subsequently, we illustrate how to use these formal definitions to represent our concrete example and the desired Web service application that could be generated from this example. **Chapter** 5 presents the proposed approach for opening Web applications for third-party development. It comprises a general picture of the approach as well as detailed explanations on the used concrete example. All activities that have to be performed by the developer to generate individual Web services are detailed in this chapter. **Chapter** 6 presents the Web service composition approach. It describes via an example and an algorithm the details about the automatic generation of Web service choreographies starting from the collaborations that exist between the generated Web services. After that, it introduces thought a set of procedures and a concrete example the details about the step of the automatic generation of Web service orchestrations starting from end-users navigation in the Web interfaces of the Web application. **Chapter** 7 introduces our service architecture recovery approach for extracting behavioral and structural models starting from the source code of the application.

This chapter describes the proposed transformation rules, and shows how they can be used to recover BPMN and SCA models starting from two examples of service oriented systems. **Chapter** 8 describes the architecture as well as the functionalities provided by two developed tools: WSGen and ArchGen, which realize our proposed ideas. In each tool, we present how the proposed approach is implemented and present the underlying technologies. **Chapter** 9 shows the details of the conducted experimentation in order to evaluate the proposed ideas. It describes the used measures to show the effectiveness and the applicability of the proposed solution. This chapter ends with discussing the threats to validity. Finally, in **Chapter** 10, we conclude the dissertation and we draw some perspectives.

Related publications

1). Mohamed Lamine Kerdoudi, Chouki Tibermacine, and Salah Sadou. Opening web applications for third party development: a service-oriented solution. Accepted in the in Service Oriented Computing and Applications journal, (**SOCA**), pages 1–27, February 2016, Springer.

2). Chouki Tibermacine and Mohamed Lamine Kerdoudi. Migrating component-based web applications to Web services: Towards considering a "Web interface as a service". An extended abstract published in proceedings of "**Journées GDR - GPL - CIEL - AFADL**" Nancy 2013. Invited by the working group **COSMAL** to present the **ICWS'12** paper.

3). Chouki Tibermacine and Mohamed Lamine Kerdoudi. Migrating component-based web applications to Web services: Towards considering a "Web interface as a service". In proceedings of the 10th IEEE International Conference on Web Services (**ICWS'12**), editors Carole A. Goble, Peter P. Chen, and Jia Zhang, pages 146–153, Honolulu, Hawaii, USA, 2012. IEEE Computer Society. **Acceptance rate: 17%**.

4). Chouki Tibermacine and Mohamed Lamine Kerdoudi. Migration d'applications à base de composants Web en services et orchestration de services Web. In proceedings of the french-speaking conference on Software Architectures, (**CAL'11**) Lille, France.

5). Chouki Tibermacine and Mohamed Lamine Kerdoudi. From web components to Web services: Opening development for third parties. In proceedings of European Conference on Software Architecture (**ECSA'10**), editors Muhammad Ali Babar et Ian Gorton, volume 6285 de Lecture Notes in Computer Science, pages 480–484, Copenhagen, Denmark, 2010. Springer.

Part I

State of the Art

CHAPTER 2

Background

Education is what remains after one has forgotten what one has learned in school.
Albert EINSTEIN.

2.1 Introduction

In this chapter, we give a background about the context of this work, which helps in the understanding of the concepts that are used in this thesis. We begin (in Section 2.2) with an overview about the Web component-based application development. Subsequently, we introduce the Web Framework features, the Java Entreprise Edition Platform, and the components of Java EE multi-tiered applications. In Section 2.3, we discuss the service oriented development from a conceptual and a technological point of views. We present in particular the Web service technology and their composition. At the end, we introduce briefly the BPEL and BPMN languages, and the SCA specification.

2.2 Web Component based Application Development

Web component-based development has emerged as a new solution that emphasizes the separation of concerns by decoupling Web application into customizable and reusable software entities called Web components. In this section, we give a brief description of Frameworks that support the Web component based development. After that, we introduce the Java Enterprise Edition Platform and its Web components.

2.2.1 Web Application Frameworks

Nowadays, there are several Web Application Frameworks that offer a high level of flexibility for the combination of different technologies to support developing Web Component-based applications. Many of these Frameworks provide libraries for database access, templating frameworks and session management, and they often promote code reuse. The Model-View-Controller (MVC) pattern is commonly used as the underlying architectural pattern for most of the Web application Frameworks. The MVC pattern is an architectural pattern originally developed for Smalltalk, an object-oriented programming (OOP) language. It enables the clean separation of the presentation logic, control logic, and business objects. Examples of Web application Frameworks that support the MVC pattern are: (i) **Struts Framework**: Struts is an open-source Web application Framework for developing Java EE Web applications [Franciscus et McClanahan, 2002]. It follows the MVC design paradigm and uses JEE technologies such as Servlets, JavaServer Pages and JSP tag libraries. (ii) **JavaServer Faces (JSF) Framework**: JSF is a Java specification for building component-based user interfaces for Web applications [Burns et Kitain, 2009]. The JSF framework is responsible for interacting with client, and it provides tools for tying together the visual presentation, application logic, and business logic of a Web application [Geary et Horstmann, 2004]. (iii) **Django Framework**: Django is a high-level Python Web Framework that encourages rapid development and elegant design[Holovaty et Kaplan-Moss, 2009]. Django's primary goal is to ease the creation of complex, database-driven Web applications. The core Django Framework is based on the MVC architecture. It consists of an object-relational mapper (ORM) that mediates between data models (defined as Python classes) and

a relational database (“Model”), a system for processing HTTP requests with a Web templating system (“View”), and a regular-expression-based URL dispatcher (“Controller”).

2.2.2 Introduction to Java Enterprise Edition Platform

The aim of the Java Platform Enterprise Edition (Java EE) platform is to provide to developers a Framework for developing and deploying Java Web component based applications, while shortening development time, reducing application complexity, and improving application performance.

Distributed Multi-tiered Applications

The Java EE platform uses a distributed multi-tiered application model for enterprise Web applications[Oracle,]. The application logic is divided into different kind of components. These components make up the Java EE application are installed on various machines depending on the tier in the multi-tiered Java EE environment to which the application component belongs. Figure 2.1 shows two multi-tiered Java EE applications divided into the tiers described in the following list:

- Client-tier components run on the client machine.
- Web-tier components run on the Java EE server.
- Business-tier components run on the Java EE server.
- Enterprise information system (EIS)-tier software runs on the database server.

The Java EE multi-tiered applications are generally considered to be three-tiered applications because they are distributed over three locations: client machines, the Java EE server machine, and the database or legacy machines at the back end.

Packaging and Deployment

A Java EE application is packaged into one or more standard units for deployment to any Java EE platform–compliant system. Each unit contains a set of components, such as an enterprise bean, Web page, Servlet, or applet, and an optional deployment descriptor that describes its content. A Java EE application is delivered in a Java Archive (JAR) file, a Web Archive (WAR) file, or an Enterprise Archive (EAR) file. A WAR or EAR file is a standard JAR (.jar) file with a (.war) or (.ear) extension. Each of these archives has a specific structure.

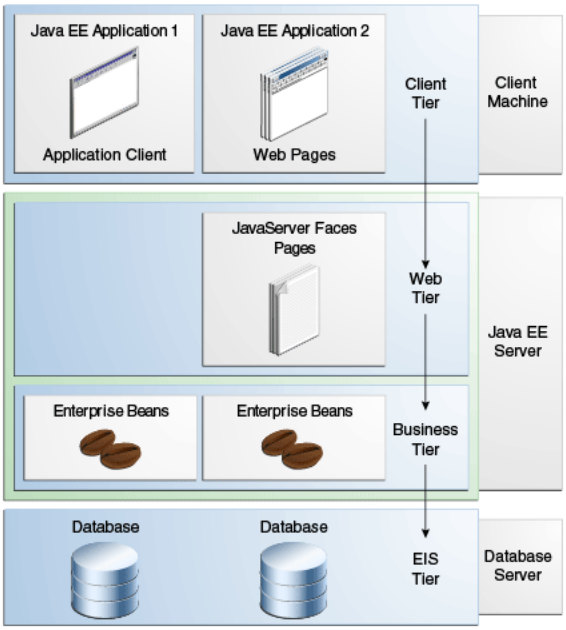

Figure 2.1 : Multi-tiered Architecture for JEE Component based application[Oracle,]

Containers

Containers are the interface between a component and the low-level, platform-specific functionality that supports the component. They provide components with services such as life cycle management, security, deployment, and threading. All JEE components are deployed into Java EE containers. The server and containers are as follows:

- Java EE server: is the runtime portion of a Java EE product. A Java EE server provides EJB and Web containers.
- EJB container: it manages the execution of enterprise beans for Java EE applications.
- Web container: it manages the execution of Web components and some EJB components for Java EE applications.
- Application client container: it manages the execution of application client components.
- Applet container: it manages the execution of applets. It consists of a Web browser and a Java Plug-in running on the client side.

2.2.3 Java EE Components

Java EE supports the development of components corresponding to each level (tier) of the multi-tiered architecture.

Java EE Clients

A Java EE client is usually either a Web client (sometimes called a thin client), an application client. A Web client consists of two parts: (i) Dynamic Web pages containing various types of markup language (HTML, XML, and so on), which are generated by Web components running in the Web tier, and (ii) A Web browser, which renders the pages received from the server. An application client (such as: Applets, JavaBeans,...) runs on a client machine.

Web Components

Java EE Web components are either Servlets or Web pages created using JavaServer Faces technology and/or JSP technology (JSP pages) and run in the Web tier. Servlets are Java programming language classes that dynamically process requests and construct responses. JavaServer pages are like regular HTML pages with special tags and execute as Servlets. JavaServer Faces technology builds on Servlets and JSP technology and provides a user interface component framework for Web applications.

There are three main types of JSP constructs: scripting constructs, directives, and actions [Falkner et Jones, 2004]. Scripting elements are used to specify Java code that will become part of the resultant Servlet. Directives are used to control the overall structure of the resultant Servlet. Actions are used to control the behavior of the JSP engine. There are three types of JSP scripting constructs that can be used to insert Java code into a resultant Servlet: *expressions*, *scriptlets*, and *declarations*

- A JSP *expression* is used to insert a Java expression directly into the output. It has the following form:

  <%= Java expression %>

  The expression is evaluated, converted into a string, and sent to the output stream of the Servlet.

- A JSP *scriptlet* is used to insert Java statements into the Servlet's `jspService` method, which is invoked by the service method. A JSP *scriptlet* has the following form:

  <% Java statement %>

- A JSP *declaration* is for declaring methods or fields into the Servlet. It has the following form:

  <%! Java declaration %>

JSP *expressions* and *scriptlets* can use a set of predefined variables (also known as JSP implicit objects) from the Servlet environment namely, **request** (which is an instance of

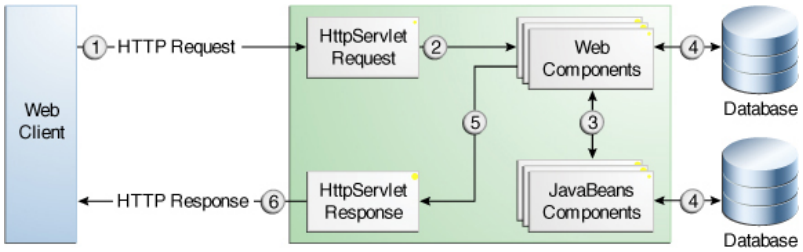

Figure 2.2 : Interaction between a Web client and a Web application that uses Web components [Jendrock *et al.*, 2014]

`HttpServletRequest` class), **response** (which is an instance of `HttpServletResponse` class), **out** (which is an instance of `PrintWriter` class), **session** (represents the `HttpSession` object), etc.

The Servlets and the JSPs are Java scripts that execute at the server-side. Figure 2.2 illustrates the interaction between a Web client and a JEE Web application that uses Web components. The client sends an HTTP request to the Web server. A Web server that implements the Java Servlet and JavaServer Pages technology (such as `Apache Tomcat`) converts the request into an `HTTPServletRequest` object. This object is delivered to a Web component, which can interact with JavaBeans components or a database to generate a dynamic content. The Web component can then generate an `HTTPServletResponse` or can redirect the request to another Web component. A Web component eventually generates a `HTTPServletResponse` object. The Web server converts this object to an HTTP response and returns it to the client.

Business Components

Enterprise JavaBeans (EJB) components are business components that run on the business logic server. The EJB components use the JavaBeans conventions for defining accessor methods for their properties.

2.3 Service Oriented Development

Service-oriented computing represents a new generation of distributed computing platform. In this section, we first analyze the nature of Service Oriented Architecture, and we present a clear definition of the concept of a service. Subsequently, we illustrate the characteristics of Web services and service composition which lays the foundation for creating Web service oriented systems.

2.3.1 Service Oriented Architecture (SOA)

The Service-Oriented Architecture (SOA) is an effective response to the problems faced by organizations to improve flexibility and reduce the maintenance's cost of their business processes, while reducing the burden of IT on the overall organization [Erl, 2009]. Service-Oriented Architecture is a concept and an approach for developing distributed architectures centered on the notion of a service relationship between applications and the formalization of this relationship in a contract. Figure 2.3 encapsulates the idea of SOA [Sommerville, 2011]. Service providers design and implement services and specify the interface to these services (as WSDL descriptions in case of Web services). The interfaces of these services are published by the service providers in an accessible registry (such as UDDI). Service requestors (sometimes called service clients) who wish to make use of a service discover the specification of that service and locate the service provider. They can then bind their application to that service and communicate with it, using a communication protocol (such as the standard SOAP).

2.3.2 Service-orientation design principles

Eight service-orientation design principles are defined by Thomas Erl's [Erl, 2008] as follows:

- **Standardized Service Contract**: Services within the same enterprise or domain are in compliance with the same contract design standards.
- **Service Loose Coupling**: Service contracts impose low consumer coupling requirements and are themselves decoupled from their surrounding environment.
- **Service Abstraction**: Service contracts only contain essential information and information about services is limited to what is published in service contracts.
- **Service Reusability**: Services have the potential to be reused. These reusable services are designed in a manner so that their solution logic is independent of any particular business process or technology.
- **Service Autonomy**: Services exercise a high level of control over their underlying runtime execution environment.
- **Service Statelessness**: Services minimize resource consumption by deferring the management of state information when necessary.
- **Service Discoverability**: Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted.
- **Service Composability**: Services are effective composition participants, regardless of the size and complexity of the composition.

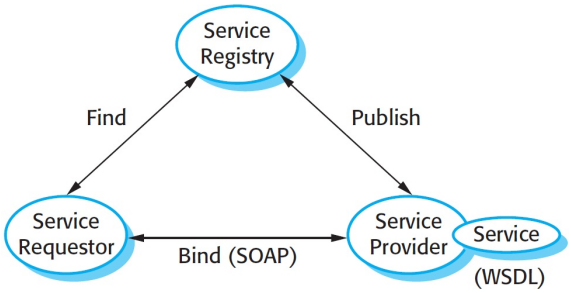

Figure 2.3 : Service Oriented Architecture [Sommerville, 2011]

2.3.3 Concept of Service

A service is a unit of solution logic (such as the "Purchase Order" service) to which service orientation has been applied to a meaningful extent. It is the application of service-orientation design principles that distinguishes a unit of logic as a service compared to units of logic that may exist solely as objects or components [Erl, 2009].

2.3.4 Service implementation technology

SOA represents an architectural model that is independent to any technology platform. In this way, the entreprise can continually pursue the strategic goals associated with SOA by continuing in taking advantage of technological advancements. A service can be built and implemented as a (SOAP)-based Web service or a REST service.

Service as Web Service

Web services technology represents now the most important and popular means for implementing service-oriented architectures. A service can be defined as: *"A loosely-coupled, reusable software component that encapsulates discrete functionality, which may be distributed and programmatically accessed. A Web service is a service that is accessed using standard Internet and XML based protocols.[Sommerville, 2011]."*. A Web service is a body of solution logic that provides a service contract consisting of a WSDL definition and one or more XML Schema definitions and also possible WS-Policy expressions. The Web service contract exposes public capabilities as *operations*, establishing a technical interface but without any ties to a proprietary communication framework.

Service as REST service

Representational State Transfer (REST) provides a means of constructing distributed systems based on the notion of resources. REST is a software architectural style inspired from the Web architecture. The term representational state transfer was introduced and defined by `Roy Fielding` in his doctoral dissertation [Fielding, 2000]. REST services (or RESTful Services)

are programs that are designed with an emphasis on simplicity, scalability, and usability [Erl, 2009]. They can send messages without a SOAP envelope and in a free encoding (XML, JSON, binary, plain text). The Web Application Description Language (WADL) [Hadley, 2009] is an XML-based language which can be used to describe the REST services. A WADL description comprises essentially the following elements:

- *Resource*: it describes a set of resources, each identified by a URI.
- *Method* : it describes the input to and output from an HTTP protocol method (GET,POST, PUT, DELETE) that may be applied to a resource.
- *Request*: It describes the input to be included when applying an HTTP method to a resource.
- *Response*: It describes the output that results from performing an HTTP method on a resource.

WADL was submitted to the World Wide Web Consortium by Sun Microsystems on 2009, but, it is not yet standardized [Hadley, 2009].

2.3.5 Web service: Standard languages and protocols

Web services refers to a collection of standards that cover interoperability. These standards define both the protocols that are used to communicate and the format of the interfaces that are used to specify services and service contracts [Josuttis, 2007]. There are five fundamental Web services standards: XML, HTTP, WSDL, SOAP, UDDI. The two first standards are existed beforehand and were used as a basis to realize the Web services approach.

1. **Extensible Markup Language**: The XML[Bray *et al.*, 1998] is used as the standard format to describe models, formats, and data types. All Web services standards are based on XML.
2. **Hypertext Transfer Protocol**: HTTP (including HTTPS) is the low-level protocol used by the Internet. HTTP(S) is one possible protocol that can be used to send Web services over networks, using Internet technology.
3. **Web Service Description Language**: The Web Service Description Language (`WSDL`) is an XML-based standard language [Chinnici *et al.*, 2007], which is used to describe Web service interfaces. It allows exposing, the functionality of a Web service as a set of operations and messages (operation parameters). In the following, we describe the different parts of the `WSDL` document.

- `Type`: is composed of type definitions which are described using a type system such as XML schemas.
- `Message`: describes the name and the type of the set of data being communicated (Invocation parts and returned values).
- `Operation`: corresponds to an abstraction, which describes an action implemented by the service.
- `Port Type`: describes a set of operations. Each operation has zero or more messages at the input and/or the output.
- `Binding`: specifies the binding of a port type to a concrete protocol, and the data format.
- `Port`: defines an endpoint as a combination of binding and network address.
- `Web Service`: is a collection of related endpoints. It specifies also the service name.

These elements can be developed as separate and then be reused or combined to form the complete WSDL document. The WSDL documents are defined by the service providers as service contracts. Applications can access functionalities exposed by WSDL documents, by simply formulating requests, which embed XML-based (SOAP) messages. The results (answers of the requests) are returned as messages of the same kind.

4. **Communication protocol: Simple Object Access Protocol**: The Simple Object Access Protocol (`SOAP`) is a protocol of the XML family [Box *et al.*, 2000] providing a communication mechanism in a distributed and decentralized environment. It defines a common and a standard format for XML messages over HTTP and other transports. SOAP was originally defined by Microsoft and IBM, but it became a W3C recommendation on June 24, 2003. Since then, the software industry has entered a consolidation phase and has begun to broadly adopt SOAP in the context of Web Services. `SOAP` consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses. SOAP messages are XML documents that contains three elements composing a message: a mandatory SOAP envelope, an optional SOAP header, and a mandatory SOAP body. The envelope represents the message, the header is a generic mechanism for include additional features (such as security, transactions, and other quality-of-service attributes) to a SOAP message in a decentralized manner without prior agreement between the communicating parties, and the body is a container for mandatory information intended for the ultimate recipient of the message.

5. **Universal Description Discovery and Integration Registry** : The Universal Description Discovery and Integration (UDDI) provides a publicly accessible means to store and retrieve information about Web services interfaces. The UDDI specifications define an

XML-based registry, established by an industry consortium to create and to implement a directory of Web services [Clement *et al.*, 2004]. A UDDI registry service manages in standard manner, information about service providers, service implementations, and service metadata. The companies can publish the descriptions of their Web services in UDDI directory as WSDL files, so customers can easily find the Web services they need by querying the UDDI registry. When customers finds the suitable Web service, they download the WSDL document from the UDDI registry, then from the information included in the WSDL file (such as the URL the Web service and how to use it), the client can invoke the functionality of the Web service.

2.3.6 Service Composition

A service composition is a fundamental concept in service-oriented computing in which, a composite service is an aggregation of services collectively composed to automate a particular task or business process. We distinguish two concepts of service composition, the "orchestration" and "choreography" of Web services. Both concepts imply coordination or control the act of making individuals Web services work together to form some coherent overall process. Orchestration by convention refers to coordination at the level of a single participant's process, whereas choreography refers to the global view, spanning multiple participants [Havey, 2005]

Web Service Orchestration

A service orchestration is a coordination of the execution of different services under control of a single endpoint central process (which can be another Web service). In other word, a Web service orchestration is the invocation of several Web services based on a well defined business process and exposing them as a single Web service (a Web service composite). So the orchestration is centralized with explicit definitions of operations and the order of invocation of Web services. The involved Web services do not know (and do not need to know) that they are involved in a composition process (see Figure 2.4). The client in Figure 2.4 can be another Web service that invokes the of operations of the Web service composite.

Web Service Choreography

A service choreography in contrast to a service orchestration is more collaborative in nature (there is no central coordinator). Rather, each Web service involved in the choreography knows exactly when to execute its operations and with whom to interact. The choreography is represented by a message exchange between the services which are provided by a set of participants (at least two participants), in order to ensure some interoperability. Each participant is responsible of implementing decisions of their internal business process (see Figure 2.5). The services of each participant could be implemented by completely different languages or frameworks such as: Java, EJB, C♯ or BPEL. Each of these services could be implemented as an individual Web service or a Web service orchestration.

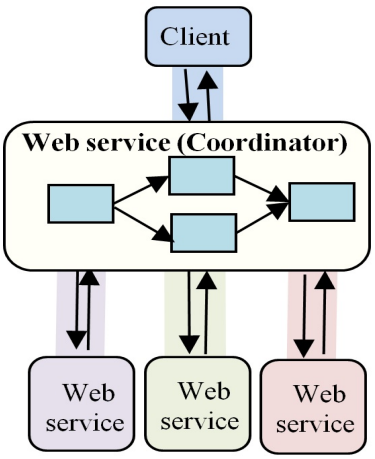

Figure 2.4 : Composition of Web services with orchestration

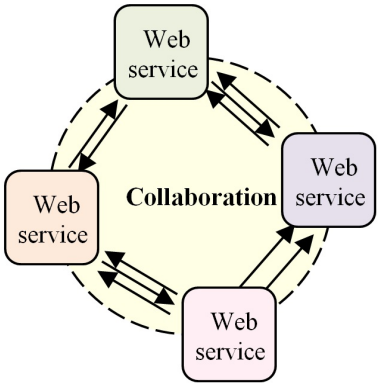

Figure 2.5 : Composition of Web services with choreography

2.3.7 Business Process Execution Language

One of the key standards accelerating the adoption of SOA is Business Process Execution Language for Web services (BPEL) [OASIS., 2007]. BPEL is an XML-based language, which is created to address the requirements of composition of Web services in a service-oriented environment. It is considered as one of leading languages for implementing Web service orchestrations. A BPEL process specifies the order in which the involved Web services should be invoked. With BPEL, we can express, sequential, parallel, conditional, and loops behaviors. For example, we can use a conditional behavior if an invocation of a Web service depends on the value of a previous invocation. We can also declare variables, copy and assign values, define fault handlers, and so on. By combining all these constructs, we can define complex business processes as an algorithm specification.

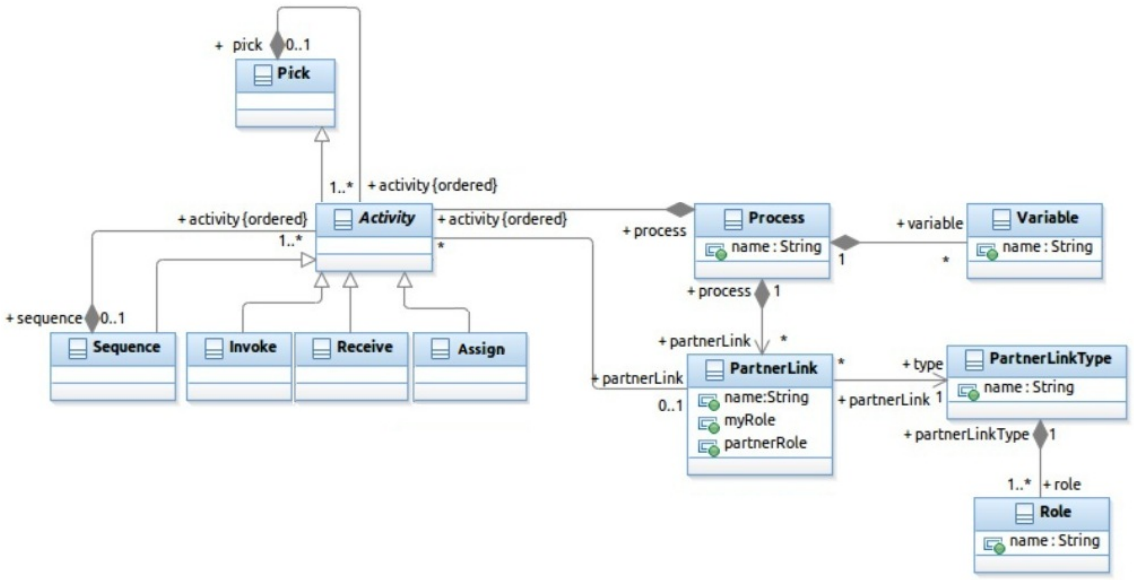

Figure 2.6 : An excerpt of the BPEL meta-model

BPEL Meta-Model

Figure 2.6 depicts an excerpt of the BPEL meta-model. The main meta-class in this meta-model is `Process`, which represents the BPEL process instances. A BPEL process is a set of steps, where, each step is called '`Activity`'. BPEL supports primitive and structure activities. Primitive activities represent basic constructs such as:

- <*Invoke*>: invoking other Web services.
- <*Receive*>: waiting for the client to invoke the business process by sending a message (receiving a request).
- <*Reply*>: Generating a response for synchronous operations.
- <*Assign*>: Manipulating data variables.
- <*Throw*>: Indicating faults and exceptions.
- <*Wait*>: Waiting for some time.

Structure activities combine these and other primitive activities to define complex business process. The most important are:

- <*Sequence*>: for defining a set of activities that run sequentially.

- <*Flow*>: for defining a set of activities that will be invoked in parallel.
- <*Switch*>: for implementing branches.
- <*While*>: for defining loops.
- <*Pick*>: for selecting one of several alternative paths.

In addition to the activity elements, a BPEL process contains other kind of elements (see Figure 2.6) such as:

- <*PartnerLink*>: the *partners* are the parties that interact with the BPEL process. They represent both a consumer of the service that is provided by the BPEL process, and a provider of a service to the BPEL process. A BPEL process declares the list of partner links it supports and, for each, which role it performs and which role its partner is expected to perform (for example: shipping provider or scheduling provider). A process can have one or more partner links.
- <*Variable*> : A variable for use in a process or a scope, with a type based on a WSDL message type, an XSD element, or an XSD basic type.
- <*PartnerLinkType*>: is a mapping of Web service port types to partner roles. Partner link types are also defined in the WSDL files of the invoked services through the WSDL extensibility element mechanism.

Structure of a BPEL Process

In Listing 2.1, we illustrate an overview of the structure of a BPEL Process in XML format. The characters that are appended to elements, attributes, and as follows: "?" (0 or 1), "*" (0 or more), "+" (1 or more).

```
<process name="NCName" targetNamespace="anyURI"
xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable">
<import namespace="anyURI"? location="anyURI"? importType="anyURI" />*
<partnerLinks>?
  <!-- Note: At least one role must be specified. -->
  <partnerLink name="NCName"
    partnerLinkType="QName"
    myRole="NCName"?
    partnerRole="NCName"?
    initializePartnerRole="yes|no"?>+
  </partnerLink>
</partnerLinks>

```

```
14 <messageExchanges>? </messageExchanges>
15
16 <variables>?
17   <variable name="BPELVariableName" messageType="QName"? type="QName"?
18           element="QName"?>+ from-spec?
19   </variable>
20 </variables>
21
22 <correlationSets>?
23   <correlationSet name="NCName" properties="QName-list" />+
24 </correlationSets>
25
26 <faultHandlers>?
27    <!-- Note: There must be at least one faultHandler -->
28 </faultHandlers>
29
30 <eventHandlers>?
31    <!-- Note: There must be at least one onEvent or onAlarm. -->
32 </eventHandlers>
33   Activity
34 </process>
```

LISTING 2.1 : Structure of a BPEL Process[OASIS., 2007]

In the Line 33, we need to define an ordered list of activities that must be executed by the BPEL process. A BPEL process starts providing services to its partners through inbound message activities (<*Receive*>, <*Pick*> and <*OnEvent*>) and corresponding <*Reply*> activities.

2.3.8 Business Process Model and Notation

BPMN is a graphical design language that can be used by business analysts or developers to represent in standard way business processes in an intuitive visual form. It provides a rich framework for modeling inter-participant processing. The BPMN specification [OMG., 2011a] define the notation and semantics of Process, and Collaboration diagrams. A Collaboration is a collection of **Participants** shown as **Pools**, their interactions as shown by **Message Flows**, and may include Processes (orchestrations) within the Pools.

The most important BPMN elements that are used in a Collaboration are:

- `Participant`: is a part of a Collaboration. It represents a specific PartnerEntity (e.g., a company) or a more general PartnerRole (e.g., a buyer, seller, or manufacturer).

- `Pool`: is the graphical representation of a Participant in a Collaboration. A Pool is a square-cornered rectangle, which has a label placed in any location within the Pool. A Participant is often responsible for the execution of the Process enclosed in a Pool. A Process generally called a workflow.
- `Lane`: a Lane in a pool represents a subdivision of the participant often a department or division of the company.
- `Message Flows`: connect two separate Pools. Message Flows cross the Pool boundary to attach to the appropriate Activity. They must not connect two objects within the same Pool. A Message Flow is a line with an open circle line start and an open arrowhead line end that must be drawn with a dashed single line
- `Sequence Flow`: is used to show the order of flow elements in a Process. Each Sequence Flow has only one source and only one target. The Activities within a Pool are organized by Sequence Flows. A Sequence Flow is line with a solid arrowhead.
- `Activity` is a basic element of BPMN. It is a step in a process that performs a work. A BPMN activity can be atomic or compound. An atomic activity, also known as a task, performs a single action. A compound activity, also known as a process (or subprocess), has its own set of atomic or compound activities, as well events, gateways and subprocess.

2.3.9 Service Component Architecture Specification

Service Component Architecture (SCA) is a set of specifications which describe a model for building applications and systems using a Service-Oriented Architecture (SOA). SCA provides a model both for the composition of services and for the creation of service components. It aims to encompass a wide range of technologies for service components and for the access methods which are used to connect them [Beisiegel *et al.*, 2009].

SCA Assembly Model

The SCA Assembly Model consists of a series of artifacts which define the configuration of an SCA Domain in terms of composites which contain assemblies of service components and the connections and related artifacts which describe how they are linked together.

SCA Component Diagram

One basic artifact of SCA is the **component**, which is the unit of construction for SCA. Figure 2.7 illustrates some features of an SCA component. A component consists of a configured instance of an implementation, where an implementation is the piece of program code providing business functions. The business function is offered for use by other components as

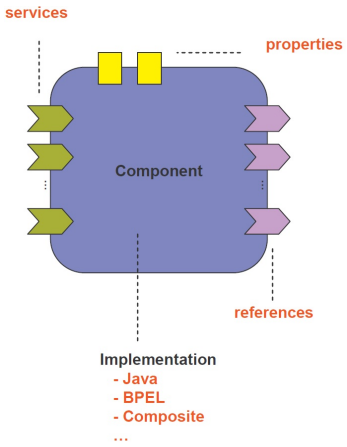

Figure 2.7 : SCA Component Diagram[Beisiegel *et al.*, 2009]

Services. Implementations can depend on services provided by other components. These dependencies are called **References**. Implementations can have settable **properties**, which are data values that influence the operation of the business function. SCA allows for a wide variety of implementation technologies such as Java, C++, and BPEL.

SCA Composite Diagram

The component configures the implementation by providing values for the properties and by linking the references to services provided by other components using **Wires**. SCA describes the content and linkage of an application in assemblies called **composites**. Figure 2.8 picture illustrates some features of a composite assembled using a set of components. Composites can contain components, services, references, property declarations, plus the wiring that describes the connections between these elements.

SCA Domain Diagram

Composites are grouped within an SCA Domain. An SCA Domain represents a set of services providing business functionalities that are belong to the same area. For example, an SCA Domain might cover all financial related function in accounts department, and it might contain a series of composites dealing with specific areas of accounting, for example, with customer accounts. The composites can be used to group and configure related artifacts. Figure 2.9 picture illustrates an SCA Domain assembled from a series of high-level composites, some of which are in turn implemented by lower-level composites

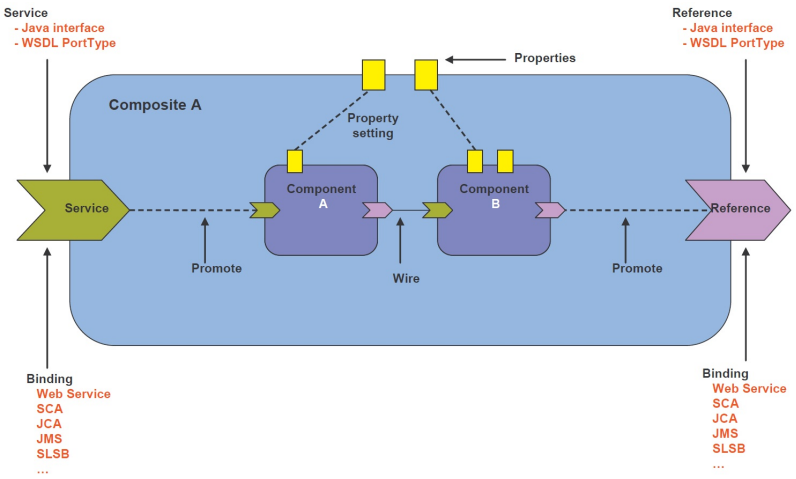

Figure 2.8 : SCA Composite Diagram[Beisiegel *et al.*, 2009]

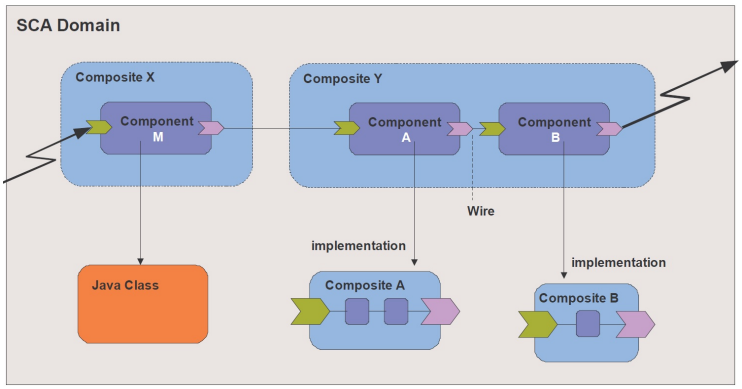

Figure 2.9 : SCA Domain Diagram[Beisiegel *et al.*, 2009]

2.4 Summary

In this chapter, we presented the main concepts related to Web component based applications and Web service oriented applications. We started by presenting the properties of the first kind of applications and their supported Web Frameworks. After that, we have introduced the Java Enterprise Edition Platform, the distributed multi-tiered application model, and the JEE components. In Section2.3, we illustrated the advantage of the service oriented development, the service oriented architecture, and the service oriented design principals. We introduced also the different implementation technologies for the primitive services and the service compositions namely, the REST services, the Web services, and the BPEL language. At the end of this chapter, we presented the BPMN language and the SCA specification which are among the leading languages for creating respectively, the behavioral and structural views of the service oriented systems.

CHAPTER 3

Literature review

We cannot solve our problems with the same thinking we used when we created them.
Albert EINSTEIN.

3.1 Introduction

In this chapter, we present a state of the art of SOA migration approaches. There are more than hundred approaches and tools for migrating systems to SOA. Each of them concentrates on specific activities in the migration and a particular kind of systems. In Section 3.2, we introduce the existing approaches and tools to migrate systems toward (Web) services-oriented applications. In this section, we distribute these approaches in four categories. In each category, we present the approaches that are the most representative and close to our contributions. In Section 3.3, we present a state of the art of the approaches in the domain of (Web) service composition. In Section 3.4, we discuss the existing approaches of software architecture recovery. In each of all the presented approaches, we discuss the similarities and differences between their proposals and our contributions.

3.2 Approaches and Tools for Migrating Systems to (Web) Services-Oriented Applications

In this section, we present the existing approaches and tools that contributed in solving the problem of migrating systems to (Web) services-oriented applications. We grouped these works in the following categories:

3.2.1 Approaches for migrating Web applications to SOA

Some works have been proposed in the literature for migrating Web applications to Service Oriented Architectures (SOA), such as [Almonaies *et al.*, 2013; Tatsubori et Takashi, 2006; Upadhyaya *et al.*, 2012; Upadhyaya *et al.*, 2015; Han et Tokuda, 2008; Baumgartner *et al.*, 2004; Lorenzo *et al.*, 2007; Belushi et Baghdadi, 2007; Djelloul *et al.*, 2009; Jiang et Stroulia, 2004; Guo *et al.*, 2005; Sosa *et al.*, 2013].

The authors of [Almonaies *et al.*, 2013] proposed a Framework which allows a semi-automatic migration of Legacy Web applications (implemented using the PHP scripting language) to service oriented applications. The proposed Framework uses an iterative process of incremental steps to analyze and reprogram existing Web applications to Web services based on the Service Component Architecture (SCA) Web services standard. The first step in this process is the service identification. The output of this step is a marked-up version of the Web application source code in which sections of code with the desired business functionality have been identified as operations of the candidate service. This tagged candidate service is then the input to an automated migration process. The migration process extracts and separates the identified business features in dynamically-typed scripting languages as object-oriented classes. An automatic inferring of the types of parameter values in dynamically-typed scripting languages is performed. At the end of the process, each object-oriented class is converted into an SCA service component.

In contrast to our approach, the authors of [Almonaies *et al.*, 2013] focus on the service migration aspect, while the identification of services is done manually based on the developer knowledge. In the identification notation, each candidate service operation is marked up manually using a <service function = function-name> tag, where function-name is a user-suggested name for the candidate service operation. In our work, we deal with the service identification task in an automatic manner. We parse automatically programs executed at sever side in the aim of identifying the potential operations to be published as Web services. In addition, we deal with many aspects that are not covered in the work of [Almonaies *et al.*, 2013] such as: i) generating service compositions starting from the dependencies between the Web components of the application. ii) taking into consideration in the service identification many aspects namely: session variables, cookies, client-side scripts, in-line documents.

The proposed work in [Tatsubori et Takashi, 2006] addresses the problem of extracting Web services from Web applications. This work proposes a model for decomposing and abstracting a Web application into modular building blocks forming the desired Web services. The decomposition model is created by modeling each human transition from page to page as modular pieces of the entire service. The abstract model consists of argument passing, data extraction and context propagation in each transition from page to page. Using these two models and a set of configuration files created manually by the developer, Web service wrappers can be created for the Web applications. The decomposition technique in [Tatsubori et Takashi, 2006] do not work *a priori* with Web applications that use techniques for retrieving remote data asynchronously (like AJAX). In our approach, we deal with this kind of applications by transforming the scripts at client-side into Web service requests, while the executed program at server side is exported as a Web service. In addition, the page transition approach in [Tatsubori et Takashi, 2006] is applied to traditional Web pages, where the navigation is done with hyperlinks contained in pages. Actually, modern Web applications that use navigation documents to implement dynamic transitions from page to page (such as in JSF Framework) are not addressed.

In the proposed approach in [Upadhyaya *et al.*, 2012 ; Upadhyaya *et al.*, 2015], RESTful services can be extracted from Web applications. The approach is based on the capturing of scenarios executed by a user for the task to be migrated as a RESTful service. The input, output and HTTP methods of the task are identified by the analysis of the annotation logs, the execution logs, and the request/response HTML Web pages.

Comparing to our approach, we create Web services from the source code of the Web applications, while [Upadhyaya *et al.*, 2012 ; Upadhyaya *et al.*, 2015] proposed a black box approach (without accessing to the source code) for identifying Web services from Web applications. In addition, our migration approach is a semi-automatic process, while in [Upadhyaya *et al.*, 2012 ; Upadhyaya *et al.*, 2015] the user is involved almost in every stage of their approach such as: executing scenarios, identifying HTTP methods and the input/output messages which are complex tasks.

In another work, Wike [Han et Tokuda, 2008] generates virtual Web services by extracting 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. These works are complementary solutions 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 our tool WSGen.

Baumgartner *et al.* [Baumgartner *et al.*, 2004] proposed a process to transform the unstructured data in the "front-end" of Web applications into structured data which are accessible from Web services that are generated automatically. In this way, they enabled the integration of "front-end" contents of several Web applications. The proposed process performs the "front-end" integration by exploiting existing Web-based interfaces of the Web applications to be integrated. The Web-based interfaces (for example, Web forms) are accessed and extracted automatically and the response document is translated from HTML(unstructured data) to XML (structured data) which later can be automatically processed. The generated Web service is like a wrapper program that queries the generated XML data. The activities of accessing, information extraction and their transformation to a structured data are performed using the Lixto Visual Wrapper tool [Baumgartner *et al.*, 2001]. This tool is particularly well-suited for visual and interactive creation of HTML to XML wrappers. The Lixto wrappers are embedded into an information processing Framework, called Lixto Transformation Server [Herzog et Gottlob, 2001] which enables application developers to format, transform, integrate, and deliver XML data to various devices and applications. The Transformation Server allows designers to publish the extracted information from the Web-based interfaces as Web Services that contain one or several operations. After that, it generates their WSDL documents. However, the Lixto wrapper enables developers to create stable wrappers programs to deal with changes that can occur on the Web page contents. The Lixto informs the Web service providers to react to any change in the Web page format.

The process published in [Baumgartner *et al.*, 2004] is complementary to our proposal. Indeed, in our approach, we do not extract the content from the "font-end" of the Web application to make them accessible as Web services. But, we generate Web services starting from the functionally (by the parsing of the server side scripts and back-end components of the application) provided the Web user interfaces ("font-end" of the Web application). The generated Web services are independent applications, any change made in the format of the Web user interface have no influence on the generated Web services.

Another work published by [Jiang et Stroulia, 2004] relies on an automated reengineering method for the presentation layer in order to create Web services for the functionalities already offered by existing Web sites. These functionalities can be specified in terms of WSDL Web-service specifications. The created Web services are deployed through proxies accessing the original Web server and parsing its responses. Indeed, the proposed process examines a multitude examples of interactions between client browsers and Web servers in order to identify a set of HTTP request and HTML response pairs. These pair-wise interactions correspond to the input and output messages of the created Web-service operation. A set of patterns are extracted from the HTML responses. These patterns are then visualized and examined by a developer in order to identify which ones correspond to useful services. At the end, the selected patterns are translated into the corresponding WSDL specifications.

Lorenzo *et al.* [Lorenzo *et al.*, 2007] proposed a wrapping based-migration approach for migrating a user-oriented interface of the Web application into a programmatic interface that exposes the full functionality and data of the application as Web services. The approach uses a black-box (without accessing or changing the source code) reverse engineering technique for modeling Web application user interface using a Finite State Automaton (the detail of this technique is addressed in [Canfora *et al.*, 2008]). The automaton model will be then interpreted by the wrapper. The wrapping technique is based on a migration process that includes the following steps: 1) Selection of the Web application functionality to be turned into a Web service, 2) Reverse engineering of the Web application user interface to identify the execution scenarios, 3) Designing of an interaction model and developing their XML-based specification to be interpreted by the wrapper, 4) Deployment and validation of the Wrapper service. This step includes all the activities needed to publish the service and export it to an Application Server. Comparing the approach in [Lorenzo *et al.*, 2007] to our migration approach, we follow a white-box approach. Indeed, we create Web services with accessing to the source code of the Web applications. In addition, our approach is based on static analysis of the system artifacts, while [Lorenzo *et al.*, 2007] analyze the system at runtime by executing scenarios which are corresponding to a set of use cases. Analyzing the system at runtime in order to identify new Web services is one of the perspectives of this work.

[Belushi et Baghdadi, 2007] proposed to wrap the functionalities of legacy Web applications to expose them as Web services. They apply a Bottom-up methodology to generate wrappers using the provided tools by J2EE and .Net environments. The wrapper program is responsible for converting the SOAP request and response messages into a format processable by the Legacy application. Another scenario provided by this approach when the Legacy application needs to invoke an external Web service. The Legacy code saves their requests in a shared file accessible by the wrapper. The later converts them into SOAP messages and invokes the desired Web services, the results are then saved in a new file that is accessible by Legacy application.

WA2WS [Djelloul *et al.*, 2009] is a Framework that can be used for creating Web Services

from existing Web applications. The creation of Web services is performed through two approaches, a reverse-engineering approach and a forward engineering approach. In the first approach, an UML conceptual schema is recovered starting from the HTML Web pages of the application using a domain ontology. The second approach uses a set of mapping rules to generate the Java source code of the Web services. In our approach, the semantic of the Web interface content is not taken into consideration when we parse the Web application. Our parsing is based on a syntactic analysis. The use of a domain ontology in the parsing of the Web application is an important complementary work that can enhance the service identification step in our approach.

[Guo *et al.*, 2005] proposed a white-box reverse engineering approach for generating wrapper components that make the functionalities of a Client–Server .NET application available as Web Services. The approach is supported by a tool called Web Service Wrapper (WSW) which is composed of two components: an Analyzer and a Wrapper. The Analyzer parses the source code of the Microsoft .Net application and it displays the results to the developer as a set of classes, properties, methods, parameters, and return values of the methods. The task of Wrapper is to wrap the legacy system into Web Services and generate related source code directly. Before that, the Wrapper uses a set of restrictions to eliminate the methods that should not be considered as Web services. An example of these restrictions is the methods that must be public and not abstract. In our approach the elimination of non pertinent operations is a semi-automatic task based on set of OCL constraints written by the developer and could be updated and reused easily.

Another work proposed by [Maras *et al.*, 2013] relies on the analysis of the client-side Web application code. The authors consider several behaviors which could be reused in a large number of Web applications [Maras *et al.*, 2012]. They proposed an approach to identify and extract the code which implements certain behaviors. The proposal is based on dynamic analysis, which relies on the execution of scenarios and saving the executed code (client-side code) responsible for an expected behavior. In addition to the behavior identification, the approach can extract library functionalities and identify (or delete) the code that does not implement any behavior (improve the performance). This work is complementary to our proposal as it deals with client side scripts' code. Analyzing such kind of client code is one of the perspectives of our work. For the moment, we partially deal with it in choreography creation when Ajax is used in the migrated Web application.

3.2.2 Approaches for migrating Legacy systems to SOA

According to [Bisbal *et al.*, 1999] a Legacy System is defined as any information system that significantly resists modification and evolution. Such kind of systems can cause to host organizations several problems such as: expensive software maintenance, because documentation and understanding of system details is often lacking and tracing faults is costly and

time-consuming. A lack of clean interfaces makes integrating Legacy Systems with other systems difficult. And, they are also difficult, if not impossible, to extend. Several techniques and methods have been proposed in literature to face the problem of migrating Legacy systems to service-oriented architectures (SOA). A set of classifications of these solutions have been proposed in [Bisbal *et al.*, 1999 ; Almonaies *et al.*, 2010 ; Zhang et Yang, 2004].

The works published in [Bisbal *et al.*, 1999 ; Almonaies *et al.*, 2010] classify these approaches into four categories which are: i) **replacement**, make sense to retire the application and replace it with a COTS components or a complete rewrite of the Legacy system from scratch. ii) **redevelopment**, which rewrites existing applications; iii) **wrapping**, which provides a new interface to a component, making it more easily accessible by other software components; and iv) **migration**, which moves the Legacy system to a more flexible environment, while retaining the original system's data and functionality. In the migration strategy, Legacy code is identified, decoupled, and extracted using approaches similar to those used in wrapping and redevelopment.

Zhang *et al.* [Zhang et Yang, 2004] classify the SOA migration approaches on: i) **black-box** re-engineering technique, is corresponding to the wrapping approach, where a set of adaptors are developed to wrap the Legacy code and data, which allow the application to be invoked as a service. ii) **white-box** approach (redevelopment) is a reverse engineering technique to the existing code in order to recover the business logic and then apply modifications on this code in order to expose this business logic as a Web service. iii) **grey-box** re-engineering technique, combines black-box and white-box approaches for integrating not the whole Legacy system but parts of a Legacy system with valuable business logic.

The approaches of migrating (Legacy) systems to SOA solutions have been the subject of few literature reviews such as [Almonaies *et al.*, 2010 ; Khadka *et al.*, 2013 ; Razavian et Lago, 2015]. Almonaies *et al.* [Almonaies *et al.*, 2010] provided a critical review based on the four categories of SOA migration approaches: replacement, redevelopment, wrapping and migration. They study the strengths and the weaknesses of each approach aiming to assist organizations to make good decisions when undertaking a new modernization project. They demonstrate that no one approach can be applied to every situation. In order to choose, the developers have to understand the maturity, applicability, strengths and weaknesses of each of approach.

Khadka *et al.* [Khadka *et al.*, 2013] provided an historic overview, focusing on the methods and techniques used in a Legacy to SOA evolution. They developed an evaluation Framework to evaluate the Legacy to SOA evolution approaches. The Framework uses six phases that are typically related to evolution/modernization of Legacy systems. The phases are identified from common phases in popular methods from software re-engineering domain and service-oriented development methodologies such as: Butterfly Method, the Renaissance Method, the Architecture-Driven Modernization(ADM), the Service-Oriented Design and Development

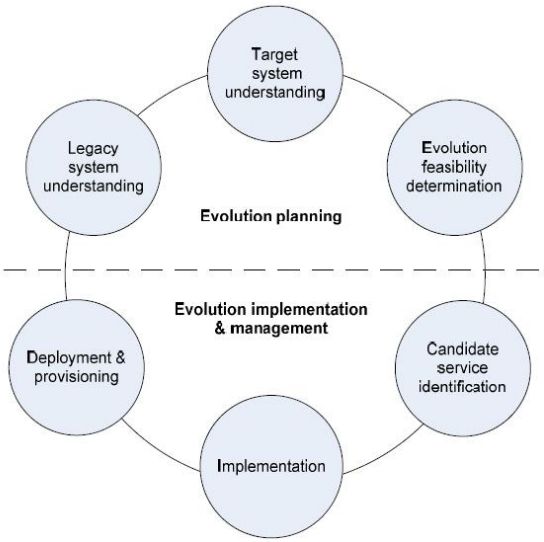

Figure 3.1 : Evaluation Framework for Legacy to SOA evolution [Khadka *et al.*, 2013]

Methodology(SODDM) and the Service-Oriented Modeling and Architecture (SOMA). Figure 3.1 depicts the evaluation Framework and its phases. The identified six phases are, **Legacy system understanding**: reverse engineering, program understanding, architectural recovery are used in this phase, **Target system understanding**: is concerned with the representation of the desired architecture of the target SOA, **Evolution feasibility determination**: The feasibility assessments are carried out at a technical, economical and organizational level, **Candidate service identification**: architectural reconstruction, feature location, design pattern recovery, cluster analysis techniques, concept analysis, source code visualization are used to identify the services in a large body of a Legacy code, **Implementation**: wrapping, program slicing, concept slicing, graph transformation, code translation, model-driven program transformation, screen scraping, code query technology, graph transformation are used to extract/leverage the Legacy code as services, and **Deployment and provisioning**: is concerned with deployment and management of the services. Based on these identified phases, [Khadka *et al.*, 2013] have derived a list of evaluation criteria that are used in their review. Indeed, a set of evaluation questions are proposed as evaluation criteria for each phase. An example of the asked evaluation questions for the **Candidate service identification** phase is : *Is there tool support for candidate service identification?* Khadka *et al.* have discussed a set of findings and good practices could be performed in each phase of this Framework.

Razavian et al. [Razavian et Lago, 2015] provided a systematic literature review of SOA migration approaches that are proposed by the research community. The authors proposed a SOA migration frame of reference which could be used to select and to define an approach for migrating to SOA. This frame of reference generalizes the activities carried out and cate-

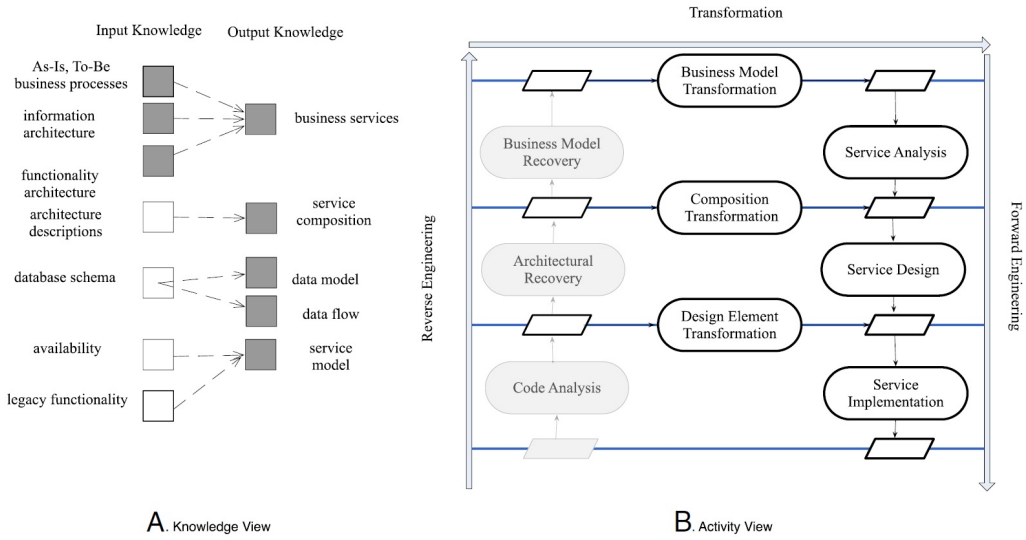

Figure 3.2 : An example of two-view approach representation. (A) Knowledge view and (B) activity view [Razavian et Lago, 2015]

gories of knowledge elements used or produced in the migration approaches. To categorize the migration approaches, the authors represent them using views, where, the view is a partial representation of a migration approach from a particular concern. The SOA migration approaches in this review, are represented using two views: i) **Knowledge view**: highlights the type of knowledges that shape and drive the migration. It concerns the understanding of input and output knowledge elements of the SOA migrations. The knowledge elements are As-Is state (i.e., AS- legacy assets) and To-Be state (i.e. To-Be services) and ii) **Activity view**: reflects what activities to be performed in SOA migration. Figure 3.2 depicts an example of the two-view approach representation that is proposed in [Razavian et Lago, 2015]. In Figure 3.2.A, the available knowledge is shown in white, and the required knowledge (not available) is in gray. The arrows between knowledge indicate what input knowledge is required to create a certain output. The Figure 3.2.B, represents the generic activities covered by the migration approach and their sequencing. Three main processes are covered in this view are:i) **reverse engineering**: recovering the lost abstractions and eliciting the legacy fragments that are suitable for migration to SOA),ii) **transformation**: from legacy abstractions to service abstractions and iii) **forward engineering**: developing the target system as result of the migration process. The subject system may not yet exist, or its existing components are renovated based on transformed abstractions as well as new requirements.

Using the SOA frame of reference, Razavian et al. have classified the existing migration approaches based on similar activity views and the migration objectives and solutions. The SOA migration approaches are categorized on eight families. Figure 3.3 illustrates the schematic forms of these families, where, (F1) Code transformation family, (F2) Service identification fam-

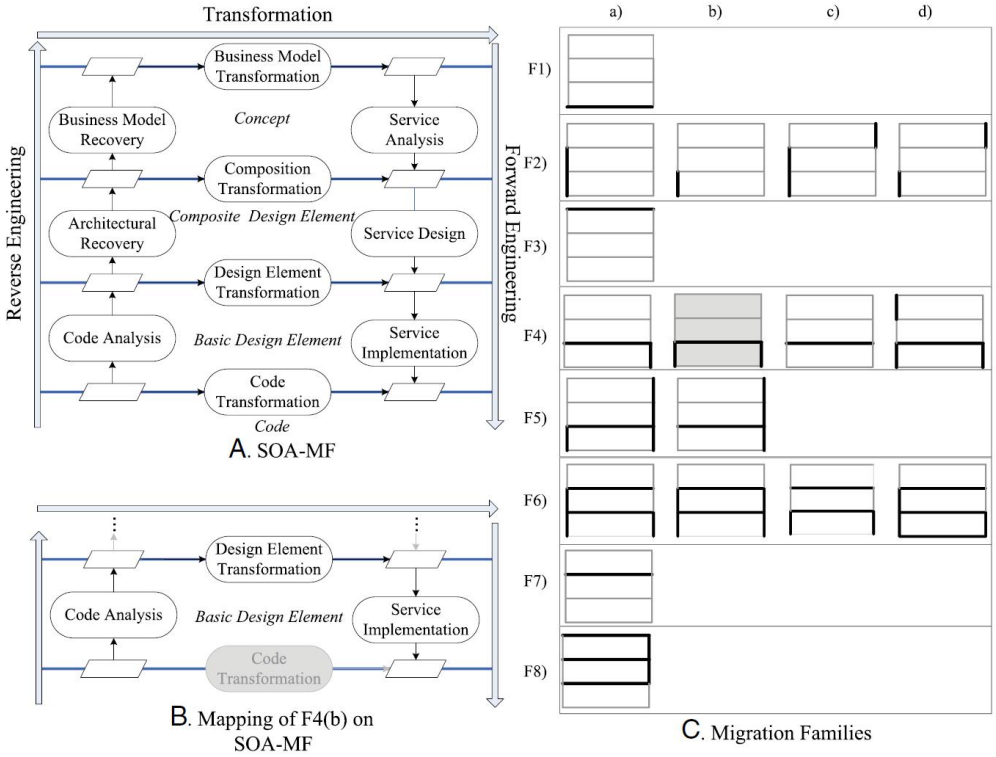

Figure 3.3 : SOA migration families [Razavian et Lago, 2015]

ily, (F3) Business model transformation family, (F4) Design element transformation family, (F5) Forward engineering family, (F6) Design and composite element transformation family, (F7) Pattern-based composition transformation family and (F8) Forward engineering with gap analysis family. Figure 3.3.B illustrates an example of the schematic form of the activity view in family (F4), where, in this family, the transformation process only occurs at "basic design element" level (e.g., modules or classes). Similarly, reverse and forward engineering processes are limited to this level. As for our approach which is published in [Tibermacine et Kerdoudi, 2010], it was categorized by [Razavian et Lago, 2015] in the family (F6). This family has a transformation at both level of "basic design element" and "composite design element"(in Figure 3.2.B). This entails altering legacy elements to services (i.e., design element transformation) as well as reshaping the structure of legacy elements to realize new service compositions (i.e., composition transformation). Migration here adopts recovering and refactoring of the legacy architecture to the service-oriented architecture as well as reshaping the legacy elements to service-based elements.

In the following, we report the representative examples of approaches for the Legacy systems migration to service-oriented architectures: The SMART approach [Lewis *et al.*, 2006]

aims at assisting organizations to migrate their Legacy systems to SOA in a systematic way. The Legacy systems functionalities, or subsets of them are exposed as services. The proposal is based on an interview guide, which is presented to the developer in terms of questions. These questions concern issues about the process of the migration. Based on the developer's responses, the degree of the difficulty and the required effort to make such migration are determined.

Sneed *et al.* [Sneed, 2006] presented a tool supported method for moving a legacy software into a service oriented solution. This work presented a set of metrics to be considered in the identification of services in Legacy systems. Three main steps are followed in this method to create Web services, which are i) **salvaging the legacy code**: consists on analyzing and evaluating several hundred of programs in order to identify and extract a code and determine if it is interesting for reusing. In this step, a domain expert is involved, and (s)he is supported by automated reverse engineering tools; ii) **wrapping the salvaged code**: the goal of the wrapping process is to provide the component extracted from the legacy code with a WSDL interface. The used technique is to transform each entry into a method and to transform each parameter into an XML data element. The tool SoftWrap has been developed to automate this transformation for the languages PL/I, COBOL, and C/C++; iii) **Linking the Web services to the business processes**: in this step, the Web services are linked as business processes. The language for implementing business processes is BPEL4WS. The BPEL4WS process establishes links to partners, defines the link types, declares the parameters to be sent and the results to be received, and invokes the Web services.

Zhang and Yang [Zhang et Yang, 2004] proposed a re-engineering approach based on hierarchical clustering algorithm to restructure the legacy code and to facilitate legacy code extraction for Web service construction. The service identification step in this approach, starts with a domain analysis to identify and document requirements on a set of systems in the same application domain. The results of this step are summarized as a domain model expressed in UML and saved in XML format. This model is used as basis to identify some business functions that are reusable and suitable to be provided as services. After that, the target Legacy system is evaluated and modeled in order to understand their source code. The modeling process uses a reverse engineering technique which is based on the use of hierarchical clustering techniques. The clustering techniques are used for transferring procedural code to object-oriented model and understanding legacy code. Clustering analysis is to group large mounts of entities in a dataset into clusters according to their relationship and similarity. In this work, the functions, procedures and classes in object oriented programs are the entities to be clustered. The obtained Dendrogram is analyzed in order to extract a functional service from legacy code according to available design decisions (using the created domain model). At the end, the extracted legacy code must be refined and packaged into candidate services. Based on the domain analysis and service identification, the service interfaces (WSDL descriptions) are designed.

Another solution has been proposed in [Canfora *et al.*, 2008] to migrate form-based Legacy systems into Web services based on a wrapping approach. In the form-based Legacy system the flow of data between the system and the user is described by a sequence of query and response interactions, which is converted into message requests from the client and message responses from the service provider. In this approach, the behaviors accrued when the user interacts with the Legacy system is modeled in terms of finite state automata, based on a black box reverse engineering technique. This specification will be interpreted by the wrapper.

Comparing these works to our migration approach, we do not create wrappers for the functionalities exposed by the Web application, but we create a new Web service application starting from the functionality exposed by the Web application. The generated Web service application will be eventually used and extended remotely by third party developers. However, the existing Web applications are kept running and accessible for end users.

3.2.3 Approaches for generating Web services from software components

A little work has been done on the migration of software components to SOA. The proposed approach in [Lee *et al.*, 2005] provides a service-oriented architecture for component based systems. The authors have developed a model for converting functionalities implemented in software components into Web services. This work allows 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. As an answer to this request, a Web service or a composition of Web services is generated automatically for the desired functionalities and returned to the client. The proposed model allows significantly to reduce execution at the client side, and increased the efficiency of Web servers by deploying as Web services only components that are requested by the client.

A. Marinho *et al.* [Marinho *et al.*, 2009] proposed a similar approach to [Lee *et al.*, 2005]. In addition, the proposal allows the generation of services starting from components, which are written in different programming languages. Compared to these reactive systems, WSGen is proactive. Indeed, in our approach we do not react to a client request, but we propose to the developers of the Web application to anticipate the export of some functionalities as Web services. In this way, third party developers can make remote extensions of the services exposed by the interfaces of the Web components. However, in [Lee *et al.*, 2005 ; Marinho *et al.*, 2009], only business functionalities implemented in software components are transformed. In our approach, the Web interfaces, the business functionalities and the navigation between Web interfaces are converted into stateless Web services and compositions of them.

In [Fei et Wang, 2004], the authors have been presented a conceptual model of Web components. They proposed a method for composing a set of services provided by Web components using parameterized contracts. These contracts link the services in the provided interfaces of

a given component to the services of its required ones. To satisfy a given functionality when a composition is under construction, a service is included if all its required services are satisfied by the component's environment. If some required services are not satisfied, other provided services from other components are integrated. In our approach, the Web components that we deal with do not define required interfaces. They refer to industrial solutions of Web development (like Java EE). In addition, we build compositions of services as BPEL processes starting from existing Web and business logic code, while in [Fei et Wang, 2004], compositions are built starting from formal definitions (contracts) associated with some candidate services in a repository.

R. Sindhgatta *et al.* [Sindhgatta et Ponnalagu, 2008] proposed a semi-automatic approach based on information retrieval techniques for locating components realizing services in existing systems. The proposed approach based on three main steps: i) identify possible links between the description of the candidate services and the source code implementations. ii) The retrieved links are then filtered and ranked based on the static traces for each selected component. In this step, the components and their call relations are represented using a component graph where the nodes are components and the edges are call relationships. The component graph is traversed in order to filter the set of initial links retrieved, the links are ranked based on weights assigned to them. iii) The structural dependencies between the results were further used to filter the retrieved links and identify the possible technical and functional components realizing the functionality. For farther analysis, a human expert has to confirm all source code links as relevant. The work of [Sindhgatta et Ponnalagu, 2008] is complementary to our approach. Indeed, combining the vector space information retrieval model and static program analysis using component graph will contribute mainly in improving the service identification task of our approach.

3.2.4 Model-Driven Approaches for generating Web service-oriented applications

Model Driven Software Development (MDD) is a vision of software development where models play a core role as primary development artifacts [Staron, 2006]. Models are used to reason about a problem domain and design a solution in the solution domain. For creating these models, we need to define rules for automating many of the steps needed to convert one model representation to another [Brown *et al.*, 2005]. In practice there are three common model transformations [Brown *et al.*, 2005] are, i) Refactoring transformations: reorganize a model based on some well-defined criteria, ii) Model-to-model transformations: convert information from one model or models to another model or set of models, iii) Model-to-code transformations (code generation): these transformations convert a model element into a code fragment.

Many works in the literature propose model-driven techniques to generate (Web) service-oriented applications. The work of [Ameller *et al.*, 2015] provided a state-of-the-art in Model Driven Development (MDD) for SOA systems. The authors focus on what are the characteris-

tics of MDD approaches that support SOA, what types of SOA are supported and how do they handle non-functional requirements. They conducted a mapping study (is a form of systematic literature review) on approaches that have been proposed in the scientific literature related to the use of MDD in the context of SOC. The conducted study illustrated the following observations: (1) predominance of top-down transformation in software development activities; (2) inexistence of consolidated methods; (3) significant percentage of works without tool support; (4) SOA systems and service compositions more targeted than single services and SOA enterprise systems; (5) limited use of metamodels; (6) very limited use of Non-Functional Requirements; and (7) limited application in real cases.

Here we give some examples of model driven approaches for Web service-oriented development. The authors of [Bauer et Huget, 2004 ; Bauer et Muller, 2004] have proposed a conceptual methodology based on UML 2.0 for developing Web service-based systems covering the business process, Web service interface and composition parts. The conceptual methodology can be seen as a multi-steps process. In the first step, a semantic business process specification is produced using an extension of UML 2.0 activity diagrams. Next, this specification is refined into two models: i) a static model, which is essentially the service model (interfaces of Web services) which are developed using UML diagrams and is enhanced with meta-data, such as the description of pre- and post-conditions for service invocation, and with exception definitions; ii) a dynamic model, which is essentially the creation of service choreography oriented models on the message exchange level using UML 2.0. Each of these two models is described by a platform independent model and one or more platform specific models.

In [Bauer et Muller, 2004] an approach based on MDA[1] has been proposed to transform the platform independent models specified by UML 2 sequence diagrams to Web Service composition representations (platform dependent models) specified in BPEL language. This approach aims in particular to assist the developer in coding BPEL specifications. The transformation consists on mapping of sequence diagram elements to BPEL2WS elements. This mapping is defined informally (using graphical notation elements) by associating an element from the set of sequence diagram elements with one or more elements of the set of BPEL2WS elements. The informal definition of a mapping between the two representations can be automated. However, information concerning the WSDL definition of the Web service interfaces are defined manually starting from UML class diagrams.

R. Gronmo [Gronmo *et al.*, 2004] proposed a model-driven process for building Web service compositions. The WSDL descriptions are transformed 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. At the end, a

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

set of WSDL descriptions are generated for the resulting composite services. This work provides means for making forward engineering (UML to WSDL and BPEL) and reverse engineering (WSDL to UML) by specifying bidirectional transformation rules. A set of recommendation are provided to developers in order to create Web services in the area of model driven development.

In [Yu *et al.*, 2007], proposed a MDA based approach to generate automatically BPEL processes. This approach uses transformation rules for converting orchestration models (as PIMs) specified in CCA (Component Collaboration Architecture), which is part of the UML profile for Enterprise Distributed Object Computing (EDOC [OMG.,]), into BPEL specifications (as PSMs). The approach is applicable to other orchestration languages. The same orchestration model can be transformed to different specifications defined in different orchestration languages by applying different rules. The transformation rules are specified using OMG's QVT specification (Operational Mappings language is used). These rules take as input an instance of the CCA meta-model enriched by new features and constraints. A BPEL file and a WSDL file (instances of respectively WSDL and BPEL meta-models) are the output of applying these transformation rules.

[Sosa *et al.*, 2013] describe a systematic and semi-automatic process to modernize and adapt legacy Web applications to SOA solutions. The proposed approach is based on model-driven techniques to drive the process. The first step in this process is the reverse engineering of legacy systems to obtain a model-based specification from them. Service identification is the second step, where new models conform to SoaML Meta-model (SoaML is OMG specification to describe Services Oriented Architectures [OMG, 2012]) of the Web application are created, where the services are labeled. After that, a matching between SoaML models and the business process model (described in BPMN) of the company is performed. An orchestration of services based on BPEL is generated by using a model-to-text transformation that takes as input the weaving models obtained in the previous phase. Concretely, the process proposes: (i) a model-based reengineering step; (ii) services identification and generation of the services layer for a SOA, including, on the one hand, the wrapper code to interact with the legacy Web application and, on the other hand, the publication of the services layer as Web services; (iii) a business process modeling stage, to clearly describe the organization processes; (iv) an orchestration code generation phase, to align the services layer offered with the business processes.

Another model-driven approach for creating service-oriented solutions has been proposed in [Johnston et Brown, 2006]. In this work, a UML profile has been defined for software services as a design notation for expressing the design of a services-oriented solution. The service models that are expressed in this UML profile are then transformed into a specific service implementation in WSDL. The use of this UML profile for describing software oriented services provides several benefits such as providing: i) A consistent set of concepts, notations, and semantics for modeling services and service interactions. ii) A "domain-specific language" for

service modeling to support designers of SOAs as they design and reason about their solution. iii) A consistent basis for generating service realizations from the logical service model. The mapping from design to implementation is described at two levels: generation of a service model from the analysis model and generation of a WSDL description from the service model. This mapping is supported though automated tools in the IBM Rational Software Development Platform. However, the defined model transformations can be used to convert service models expressed using the profile into various target languages for service realization.

C. Dumez *et al.* [Dumez *et al.*, 2008] have introduced UML-S (UML for Services), an extension to UML 2.0, to develop composite Web services conforming to the model-driven engineering vision. It is an UML profile and guidelines to develop composite Web services according to MDE principles. In UML-S, both class diagrams and activity diagrams are used to model and specify respectively Web services interfaces and their interactions. A set of transformation rules between UML-S and low-level code (platform-specific code) are introduced. First, transformation rules from WSDL 2.0 to UML-S class diagram are introduced. Second, transformation rules to generate WS-BPEL 2.0 code from UML-S diagrams are provided. The use of this transformation rules has simplified the creation of composite Web services and has made this task more easy.

A. Fuhr *et al.* [Fuhr *et al.*, 2013] proposed a model driven approach to migrate legacy systems to service oriented architectures. Based on model-driven strategies, this approach provided an extension of IBM's SOMA method towards migration. SOMA(Service-Oriented Modeling and Architecture) is an iterative and incremental method to design and implement service-oriented systems. Four phases of the SOMA method are extended to support SOA migration. The defined extension are: i) The service identification phase has been extended by a model-driven technique to reverse-engineer legacy code into an appropriate TGraph, which enables queries and transformations to identify service candidates. The queries and the transformations are applied on models that represent different views on software systems including business process models, software architecture and programming code. ii) The service specification phase has been extended by combining the forward engineering (design of the target architecture and orchestration of services) with the reverse engineering (derive service operations and message design from legacy code) techniques. Therefore, the forward design is performed by analyzing legacy systems. iii) The service realization phase has been extended by the tasks static and dynamic analysis of legacy systems in order to understand the implementation of legacy functionality. The result of these analyses helps developers to look in legacy system to find the code to be migrated to services. iv) The service implementation phase has been extended by including graph transformations as techniques to extract legacy code and transform it into service implementations. The proposed approach was applied to the migration of functionality of GanttProject[2] towards a Service-Oriented Architecture. As result, a set of

[2]GanttProject: http://www.ganttproject.biz/

Web Services were generated whose business functionality where implemented by transforming legacy code.

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 [Johnston et Brown, 2006] 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, most of these 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.

3.3 Approaches for Web Service Composition

A summary of proposed solutions, standards and Frameworks for Web service composition is presented in [Milanovic et Malek, 2004]. [Benatallah *et al.*, 2003] proposed a set of patterns for the definition and implementation of composite services. These patterns suggest a methodology for providing a high level abstraction in the design, construction and maintenance of composite services.

A survey of automated Web service composition methods is presented in [Rao et Su, 2005]. The authors consider the automation is the possibility to generate the process model automatically, or to locate the correct services if an abstract process model is given. They proposed a framework for automatic Web services composition. This framework discusses the similarities and differences between the existing service composition methods. Figure 3.4 gives a general representation of this Framework. The `Translator` translates the specifications expressed by the participants (`Service requester` and `Service provider`) using external languages into another kind of specifications defined in an internal language that is used by the `Process Generator`. For each request, the `Process Generator` generates a plan that composes the available services in `Repository`. The `Evaluator` evaluates all plans and proposes the best to be executed by the `Execution Engine`.

Our proposed approach covers all the phases in this framework, but slightly in different way. Indeed, our system is not reactive to external specifications that come from service requester, but, the process plan is extracted starting from the parsing of the Web interface navigations. The translator in our approach, uses the extracted navigation documents and transform them into a set of executable BPEL orchestrations. The involved Web services in the composition are those that are generated from the Web application. The generated orchestrations could be then executed in one of the existing BPEL execution engines such as Apache ODE [Apache., 2013].

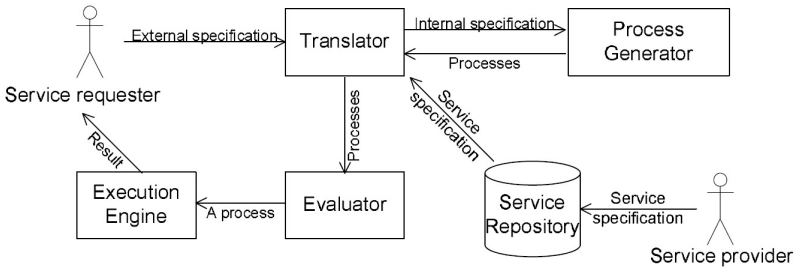

Figure 3.4 : The framework of the service composition system[Rao et Su, 2005]

The authors in [Zernadji *et al.*, 2015] proposed a scripting language called WS-BScript used by the designers to specify their process plans. These specifications are then interpreted automatically for updating BPEL orchestrations. The WS-BScript language is used mainly to reconfigure automatically an existing BPEL orchestration. The authors take into consideration the non-functional properties in the evolution of these BPEL processes.

A survey of existing methods and approaches for reliable composite services is presented in [Immonen et Pakkala, 2014]. The authors define the required phases of the composite service design and execution to achieve reliable composite service. These phases are described in the form of a framework. A matching of the existing approaches and their Framework was performed. In our Web services migration approach, we proposed to publish similar operations in different Web services, this makes them potentially published on different servers. In this way, if one of the services is not available, or does not work correctly, we can find a similar service (its substitute) in some other servers. In this way, our approach contributes in the creation of reliable composite services.

Several automatic, semi-automatic and manual service composition approaches are proposed in the literature such as [Paik *et al.*, 2014 ; Oh *et al.*, 2008 ; Segev et Toch, 2009 ; Zeng *et al.*, 2004 ; Ren *et al.*, 2011 ; Medjahed et Bouguettaya, 2005 ; Boustil *et al.*, 2014]. Paik et al. [Paik *et al.*, 2014] proposed to compose services dynamically and automatically from existing services in service-oriented architecture and cloud computing environments. The authors proposed a framework and a nested multilevel dynamic composition model which provides a functional scalability and a seamless composition. They extend the phases of automatic service composition that are proposed [Rao et Su, 2005] by adding orchestration of nested workflows and composition property transformations to the existing process.

Oh et al. [Oh *et al.*, 2008] consider the problem of automatic composition of Web services as AI planning and network optimization problems to investigate its complexity. Their Framework allows to: (i) formalize the Web service composition problem in terms of AI planning problem, (ii) analyze the available Web service sets using complex network analysis techniques, (iii) build benchmarks that simulate these observed topologies, and (iv) design a Web service composi-

tion algorithm, and finally, (v) evaluate it against the generated benchmarks as well as existing testing sets.

In [Segev et Toch, 2009] a context-based semantic approach is proposed for classifying and ranking Web services in order to compose them. The classification is based on the analysis of the WSDL documents and free text descriptions of the Web services. Two methods are used for the classification of Web services: Term Frequency/ Inverse Document Frequency (TF/IDF) and context extraction. The TF/IDF mechanism generates a set of representative keywords from a corpus of documents (WSDL files or textual descriptions). A context extraction is the process of creating a set of descriptors, where, a descriptor is a pair of terms (word, phrase, or alphanumerical) and a weight (represents the importance of the descriptor in relation with the Web service). The process of service ranking is to give a numeric estimation of the complementary relation between each two Web services. This process is based on the context analysis, where, each Web service WSDL context descriptor is evaluated according to its proximity to other services' free text context descriptors.

In our approach, some parameters of the generated operations have meaningless names such as: arg1, input0 or param1, which are given by default. Thus, the generated WSDLs for these services will have also message parts with meaningless names. This problem is considered by [Rodriguez *et al.*, 2010] as an anti-pattern (called `ambiguous names`) which occurs frequently in the Web services description. This anti-pattern makes the matching of parameters in the automatic composition of the services more difficult, and requires the developer intervention. The methods TF/IDF and context extraction that are used by [Segev et Toch, 2009] could be exploited by our tool "WSGen" to extract representative names for these parameters starting from the analysis of HTML forms and the used text to describe the input and output values in the Web application.

Medjahed et al. [Medjahed et Bouguettaya, 2005] proposed a composability model to check whether Web services can be composed without failure during their execution. A set of algorithms for checking Web service composability are provided. In this composability model, the Web services are compared to check their composability through a set of rules organized into four levels: syntactic, static and dynamic semantic and qualitative levels. Each level compares a specific pair of attributes of interacting Web services. The authors defined also, the notions of $composability\ degree$ and $\tau - composability$. The $composability\ degree$ is to associate a $weight$ for each of the four levels, and defining a $weight$ for each rule in that level. The weight is an estimate of the importance of the corresponding level or rule from the composer's point of view. The $\tau - composability$ compares the composability degree and the threshold (τ: minimum value allowed for composability degree) to decide whether an operation is partially, totally, not at all composable with another operation.

In our work, we have used the composability model of [Medjahed et Bouguettaya, 2005] to

check the syntactic composability of the services. Semantic composability is one of the perspectives of our work.

3.4 Approaches of Software Architecture Recovery

Many automated techniques and approaches have been proposed in the literature to help in recovering the architecture of a software system from its implementation. The authors in [Ducasse et Pollet, 2009] have been presented a state of the art in the software architecture reconstruction (SAR) approaches. They proposed a classification based on the lifetime of SAR approaches as the following: (i) based on the `Intended goals` such as: understanding, reuse, evolution, construction, and analysis. (ii) based on the `Followed processes` either a bottom-up (refers to a recovery process), a top-down (refers to a discovery process) or a hybrid process. (iii) based on the `Required inputs`, most SAR works are from source code constructs, but there are some other kinds of information are considered such as dynamic information extracted from a system execution, or historical data held by version control system repositories, architectural elements such as styles or viewpoints, or the fusion of multiple source of inputs. (iv) based on the `Used techniques`, they are classified on three automation levels: quasi manual, semi automatic, and quasi-automatic. (v) based on `Expected outputs`, the goals and the output are clearly related. Most of the approaches focus on identifying and presenting software architecture (such as visual software views, reconstructed architectural views, or design patterns). Some approaches provide valuable additional information such as conformance of the architecture and the implementation, or performing extra analysis on the extracted architecture to qualify it or to refine it further.

[Garcia *et al.*, 2013b] proposed a framework comprising a set of principles and a process for recovering a system's ground-truth architectures. According to the authors, a ground-truth architecture is the architecture of software system that has been verified as accurate by the system's architects or developers who have intimate knowledge of underling application and problem domain. They consider this knowledge is often undocumented. The framework's principles serve as rules or guidelines for grouping code-level entities into architectural elements and for identifying their interfaces. Four types of information are used to obtain the ground-truth architecture: generic information (e.g., system-module dependencies), domain information (e.g., architectural-style rules), application information (e.g., the purpose of the source code elements), and information about the system context (e.g., the programming language used). The proposed process recovers a preliminary version of the system architecture from its source code. After that, it involves the system's engineers in a controlled step for completing the recovery. The authors have discussed their findings through obtaining ground-truth architectures for four system existing systems come from several problem domains

The authors in [Garcia *et al.*, 2013a] have used a set of eight architectures that have been

recovered from open source systems and carefully verified as ground-truths architecture in performing a comparative analysis of six state-of-the-art software architecture recovery techniques. They have used a set of metrics to evaluate the accuracy of each technique, and their ability to identify a system's architectural components and overall architectural structure. The input of the six selected techniques in this study are: (i) textual inputs which are words in source code and comments of implementation-level entities and (ii) structural inputs which are control-flow-based and/or data flow-based dependencies between implementation-level entities. The selected software architecture recovery techniques are:

(1) Algorithm for Comprehension-Driven Clustering **(ACDC)** [Tzerpos et Holt, 2000], this technique recovers components (subsystems) using patterns driven approach. Depending on the pattern used, the subsystems are given appropriate names. These patterns are used to group entities at the implementation levels. They refer to familiar subsystem structures that frequently appear in manual decomposition of large industrial software systems. These patterns include for example grouping together entities (such as, procedures and variables) contained in the same source file (Source file pattern) and entities in the same directory (Directory structure pattern).

(2) Weighted Combined Algorithm **(WCA)** [Maqbool et Babri, 2004] is a hierarchical clustering technique based on grouping together items or entities based on their properties or features. In this technique, each entity is represented as a vector of properties (features), which represents the entity dependencies. Each implementation-level entity is placed in a cluster, where a cluster represents an architectural component. The WCA is based on an iterative merging process of clusters based on a pair-wise similarity of these clusters. The feature vector of a combined cluster is built by combining the feature vectors of the two similar clusters.

(3) ScaLable InforMation BOttleneck **(LIMBO)** [Andritsos et Tzerpos, 2005] is a hierarchical clustering technique. It differs from WCA in three points: (i) LIMBO use a mechanism called Summary Artifacts (SA) to reduce the computations needed while minimizing accuracy loss. (ii) LIMBO uses the Information Loss (IL) measure to compute similarities between entities, (iii) LIMBO associates a new feature vector for a combined cluster, which is computed using a specific formula.

(4) **Bunch** [Mancoridis *et al.*, 1999] uses a hill climbing algorithm to find a partitioning of entities into clusters that maximizes an objective function. Bunch initially starts with a random partition and stops when there is no better partition.

(5) Zone-Based Recovery **(ZBR)** [Corazza *et al.*, 2010] is a technique which utilizes textual information, hierarchical clustering, and a weighting scheme for feature vectors. The textual information tries to infer a software system's semantics when recovering the system's architecture. In ZBR, the source file divided into zones and each word in this zone is scored by a term

frequency. The zone is weighted using the Expectation-Maximization (EM) algorithm. Clusters in ZBR consist of source files and feature vectors (consists of the word's score values for each weighted zone). The ZBR computes the similarity between entities using cosine similarity.

(6) Architecture Recovery using Concerns **(ARC)** [Garcia *et al.*, 2011] recovers concerns of implementation-level entities and uses a hierarchical clustering technique to obtain hierarchical elements. The recovery of concerns is based on a statistical language model LDA (Latent Dirichlet allocation), which is obtained from the identifiers and comments in a system source code.

In our recovery approach, the components are grouped based on their names and the developer knowledge. We plan in future to improve our approach by using one of these clustring techniques. Our classification will take into consideration several other criteria such as: the cohesion, the coupling, the similarity between components.

Recently there is a major interest by researchers for proposing approaches and techniques that aim to recover component-based architectures and service-oriented architectures from the system implementations. In the following, we give some examples of these approaches:

[Chardigny *et al.*, 2008] proposed an approach called ROMANTIC which focuses on extracting component-based architectures from existing object oriented systems. The ROMANTIC approach is based on defining a correspondence model between the code elements and the architectural concepts and instantiating that model in order to extract the architectural elements from the software. The correspondence model consists on defining a partition of the system classes in shapes (each shape represents a component), while some classes in a shape have links with classes from another shape. The extraction approach is based on a hierarchical clustering algorithm. For the authors, an extracted architecture is relevant if it respects four guides which are: (i) semantically correct. (ii) a good quality properties. (iii), respects precisely the recommendation of the architect (iv) can be adapted to the specificity of the deployment hardware architecture. The authors in [Kebir *et al.*, 2012] proposed a technique for evolving object oriented systems toward component-based applications.They have used the component-based architectures that are created as a result of the ROMANTIC approach for identifying the internal structure of the software components and their required/provided interfaces from the object oriented programs.

The works in [Chardigny *et al.*, 2008] and [Kebir *et al.*, 2012] focus on creating component-based architectures which help them to identify software components and their interfaces from object oriented systems. The set of guides and the clustering algorithms are used to represent a set of classes as an architectural component element. After that, they have defined a fitness function to measure the semantic-correctness of a component, where they have identified three semantic characteristics of software components: composability, autonomy and speci-

ficity. In our work, during the identification and the creation of Web services from Web applications, we create implicitly this kind of architectural elements by clustering and distributing the services based on the cohesion, coupling, and similarity criteria. The semantic characteristics are ensured by our service distribution technique. But, the main objective of our architecture recovering approach is not to identify services from Web applications but, is to explicit architectural views from the collaboration between service participants, which can be seen at code level only. The generated architectures are then used by the maintenance developers to understand the implementation-level details of the target system.

[Allier *et al.*, 2010] proposed an approach to restructure legacy object oriented applications into component-based applications. They consider a component as a group of classes collaborating to provide a system function. The interfaces provided and required by a component are the method calls respectively from and to classes belonging to other components. The identification of components and their interfaces is based on the analysis of traces obtained by executing scenarios corresponding to the system use cases. A clustering technique is used for the classes of the target system, where the classes that appear frequently together in the execution traces are grouped to form a component. The clustering is based on the using of meta-heuristic search algorithms in order to find near-optimal solutions. The authors defined a fitness function to evaluate the quality of a partition by considering the internal cohesion of a component and the inter-component coupling.

The authors in [Seriai *et al.*, 2014a] proposed an approach to identify the interfaces of a component according to its interactions with the other components. According to the authors, a component is implemented as a set of classes that work together to provide one or more services. The set of services are then grouped as an interface for that component. They consider the interface identification as a clustering problem based on the dependencies between the exposed methods and the components that use them. The Formal Concept Analysis (FCA) techniques are used to perform this clustering. For each component a formal context is created. This context summarizes the dependencies of this component with other components. A dependency is defined by a pair: (calling component, called service). The FCA classifier module derives a concept lattice in which services are grouped according to the calling components. After that, the lattice is interpreted in order to suggest interfaces for the called component based on its corresponding lattice. The authors in [Seriai *et al.*, 2014b] proposed an approach for restructuring the object oriented systems by extracting component based architecture and provide a mapping between the architectural elements and their corresponding ones in the code. Their main objective is to identify all instances of classes representing an instance of a component in order to build the component's factory. The authors consider an instance of a component is all the class instances, which have had connections during the execution of the application. The component instances are created by transforming object call graph into a component instance call graph.

The works in [Allier *et al.*, 2010], [Seriai *et al.*, 2014a], and [Seriai *et al.*, 2014b] are complementary to our recovery approach. Indeed, in our approach, we analyze the static calls between services to create the service component architectures. The grouping of the components of these architectures is based on the developer knowledge and the syntactic comparison of the service and component names. This step of our approach could be improved by using the analysis technique of the execution traces which is proposed by [Allier *et al.*, 2010]. The grouping of the components will be then based on their runtime dependencies.

The authors in [Anquetil *et al.*, 2009] proposed an approach for making the maintainers' work easier. They proposed a component recovery approach for Java legacy application. They explored a reverse engineering approach to extract component types, data types, provided and required services, structure of composite component types, and communication channels between components. Their approach is intended to compare a concrete implementation with an abstract model. It checks also the good state of the architecture of a system by indicating when a component is used improperly (e.g. communicate with the wrong component).

The proposed approach in [Forster *et al.*, 2013] enables the identification and the analysis of software dependencies in the context of two software development frameworks: OSGi and Qt. The first step in this approach is the identification of relevant architectural styles for communication. To do so, developers are invited to inspect the available documentation to learn about predominant styles and defining the abstract communication principles. The second step consists on the parsing of the source code in order to create a model capturing object oriented and procedural language constructs (such as packages, classes, and methods). The third step is the identification of communication ports in source code. The Graphical Pattern Modeling and Matching techniques are used in this step. The last step in this approach consists on creating components and reconstructing the actual connections. The creation of component abstractions and their attached ports is supported by the system experts. The information recovered is visualized in a graphical view.

In comparison to our work, [Anquetil *et al.*, 2009] and [Forster *et al.*, 2013] recover component based architectures, where each public method in the classes is considered as a provided interface for the component that enclose these classes and each invocation to this method is modeled as a required interface for the caller component. But, our focus is to recover SOA architectures (structural and behavioral views) starting from the service-oriented applications (such as OSGi applications). Indeed, the service oriented communications are recovered and modeled. To do so, we identify the components that register services in the Service Registry, and the components that obtain from this Registry the object references (or WSDL interfaces) of these services in order to invoke them and we represent these collaborations by BPMN and SCA models.

Some works have been proposed in the literature to detect SOA design patterns from service

oriented applications such as: [Demange *et al.*, 2013], [Upadhyaya *et al.*, 2013], and [Liang *et al.*, 2006]. The SOA design patterns are cataloged in few books such as:[Erl, 2009] and [Daigneau, 2011]. These books provide good practices to design service oriented systems. In the recent years, the detection of patterns in an SOA environment starts attracting more and more researchers. Firstly, [Upadhyaya *et al.*, 2013] proposed to identify service composition patterns by analyzing the execution logs and the order of service invocation events in these execution logs. These service composition patterns represent services that are used together repeatedly and they are structurally and functionally similar. Secondly, [Demange *et al.*, 2013] proposed to detect five newly defined SOA patterns. They specify these patterns using "rule cards". According to the authors, the "rule cards" are sets of rules that combine various metrics. These can either be static which provide information about structural properties like cohesion or coupling, or dynamic, which provide information about response time or number of service invocation. Subsequently, they generate detection algorithms from rule cards and apply concretely these algorithms to detect patterns on SOA systems at runtime. [Liang *et al.*, 2006] used the property similarity between the services to detect SOA design patterns. All these works are complementary to our service oriented architecture recovery approach. Improving our approach by recovering SOA design patterns will greatly enhance the understanding of the target system.

3.5 Summary

This chapter sums up existing works in the state-of-the-art in three main categories: migration to SOA approaches, Web service composition approaches, and software architecture recovery approaches. We highlighted the focus of the community on evolving existing systems to service oriented applications. The migration to SOA has been proved to bring many benefits. Of these benefits, one most important is to open the existing systems for third party development. Through in-depth analysis of the studies focused on migration activities carried out by the selected approaches, we have shown that there are several ways to make such transition to SOA (for example, by wrapping, replacement, redevelopment, or migration), and the migration process varies depending on the type of system to be migrated (legacy system, software component, Web applications, or conceptual models) and the type of the target system (Web service, service, composition of services, service oriented models). In addition, each selected approach focuses on a set of particular activities and objectives (such as, code transformation, service identification, business model transformation, design and composite element transformation, and automatic, semi-automatic and manual migration).

Furthermore, most of the presented approaches focus on the modernization to SOA by abandoning of the existing (legacy) system. But, in our approach, we create from Web applications new service-oriented systems intended for remote extensions by third party developers, and, these Web applications are still operational and accessible by end-users via their browsers. The second category of state of the art is about the Web service composition approaches. We

have shown the topic of (semi)-automatic composition of services has attracted more and more researchers in last decades, especially in the search for reliable and semantic service composition. The third category of state of the art is about software architecture recovery. We pointed out the important role of architecture recovery in system maintenance and evolution.

The following chapters of this thesis aim at presenting the proposed ideas, which are illustrated through a set of concrete examples and evaluated with real data.

Part II

Contributions

CHAPTER 4

Formal model for Web applications and Service oriented Systems

Mathematics as an expression of the human mind reflects the active will, the contemplative reason, and the desire for aesthetic perfection. Its basic elements are logic and intuition, analysis and construction, generality and individuality.

RICHARD COURANT

4.1 Introduction

In this chapter, we introduce (in section 4.2) an example of a Web application which serves as a running example for illustrating our proposals throughout this dissertation. After that, we introduce (in section 4.3) a formal description of the context of our work, which is composed of Web applications and Web service-oriented systems. These formal definitions represent in an unambiguous way the concepts used in Web applications which are the input of our method, and the concepts used in Web service oriented systems, which are the output. Moreover, we identify the properties that characterize the concepts in both systems, and define them using a set-theoretic notation. In addition, this chapter illustrates how to use the proposed formal model to represent our example of Web application and the desired Web service oriented application that could be generated from this example.

4.2 Illustrative Example

Our example is an e-shopping Web application that offers to customers the opportunity to purchase electronic devices. The seller offers also delivery capabilities to ensure the transportation of the purchased items. The shopping process begins when the customer enters keywords for searching products via a Web interface (HTML form) provided by the application. A set of items which are relevant to these keywords are provided and distributed on HTML pages. The customer can choose one or a set of items among the returned ones. The selected items are saved in a virtual cart, and the total price is then calculated and provided via a Web interface. Once the customer finishes the shopping, (s)he is asked to sign in or to register for a new account. At the end, before proceeding to checkout, the delivery schedule is prepared and provided via another Web interface.

4.2.1 Problem Statement

In this subsection, we remind via this illustrative example the problem stated in the introduction of this thesis. Let us suppose now that a third-party developer would like to implement an extension to this Web application. This extension concerns services for the purchased items (insurance against theft, breakage, fire, etc). This extension provides first to customers an interface for searching products that are for sale by the original application. The customer selects a set of desired products and chooses the quantity for each one. The extended version of the application includes all the steps from the original version. However, it gives the opportunity to choose an insurance service and then integrate its cost to the final amount.

So, there are some functionalities needed by this extension that are already provided by the original application (eg. searching, payment and delivery schedule). Therefore, for implementing this extension it would be interesting for the third-party developer if (s)he can use

functionalities which are provided by the original application instead of implementing them from scratch.

In order to implement this solution, the third-party developer should have access to the functionality provided by the Web application differently than *via* its Web interfaces. Indeed, if (s)he uses only these Web interfaces, (s)he should send from her(his) programs the necessary HTTP requests, with customized parameters, and should then parse the returned HTTP responses. This parsing involves an analysis of the responses in order to look for some specific parts which are of interest. In our example, the third-party developer should implement a program that sends an HTTP request to the server hosting the Web application, with for example the reference(s) of the product(s) (chosen by the customer). The parsing should identify the price of the purchased items, among other elements. As stated in the introduction, this task is time-consuming, cumbersome, and error-prone. In addition, the developer should know the exact type and structure of the HTTP requests and responses.

Moreover, unfortunately, 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. Besides, 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 we cannot introduce annotations to create composition of operations which we found in real-world business logic. In addition, we cannot use these annotations to expose Web interfaces as Web services. The same observations can be made on Eclipse tools (WTP project), which allow to generate Web services starting from individual methods in Java classes.

4.2.2 Potential Web Services

As we have aforementioned in the introduction of this thesis, one of the best solutions for the previous problem is to enable the Web developer to create starting from the Web application a set of Web services suitable for remote extensions. In the presented example, the created Web services could be: a `Searching` Service for searching items, a `Cart` Service to manage a virtual shopping cart, an `Account` Service used to sign in or to register for a new account, a `Delivery` Service and a `Payment` Service.

The application extension scenario introduced previously can easily be implemented by remotely invoking the Web service operations. The extension of the shopping Web application can even be built simply as a BPEL process by invoking the `Searching` Web service using the reference of the chosen product as input. The returned price is added to the insurance's price.

The BPEL process makes the payment of the purchased products (without insurance) from

the original Web application by invoking the generated `Payment` Web service. After that, to pay the insurance costs, the BPEL process invokes an operation that is implemented by the third party developer. At the end of this process, it invokes the `Delivery` Service.

4.3 Web applications and Service oriented Systems

Before presenting the details of our approach and how Web service-oriented systems can be derived from Web applications, we define in this chapter the different concepts used in our approach. Indeed, we introduce formal descriptions of what composes Web applications and Web service oriented systems. These formal descriptions provide a better understanding of both kinds of software systems. In addition, this enables an accurate presentation in the following chapters of the processing performed on Web applications to generate Web service systems.

These two formal models have been validated in the publication [Kerdoudi *et al.*, 2016].

4.3.1 Web application Model

We adapt here the generic model of Briand *et al.* [Briand *et al.*, 1996] to represent Web applications and their elements as a directed graph expressed using a set-theoretic notation. This graph is expressed as a pair (E, R), where E symbolizes a set of software entities found in a Web application, namely, `client side artifacts` and `server side artifacts`. R is a binary relation on $E(R \subseteq E \times E)$. It corresponds to the relationships between Web application elements, representing both structural and behavioral dependencies. Figure 4.1 gives an example of such a representation for an imaginary Web application.

Definition 1 *Representation of a Web application*

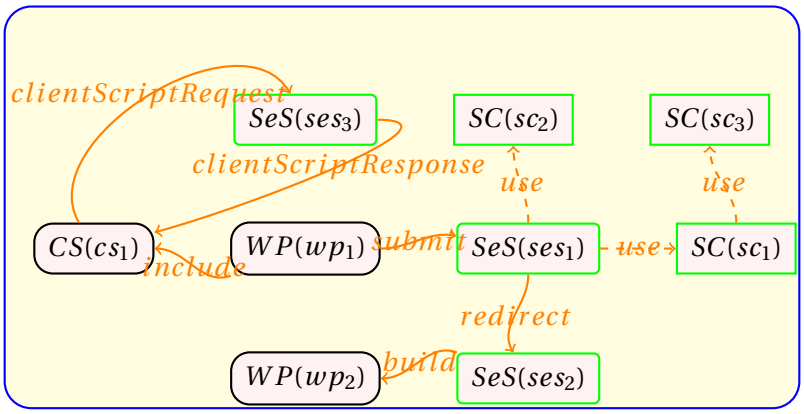

Figure 4.1 : Structure of a Web application

A Web application is represented as a pair (E, R)*, where*

- $E = CSA \cup SSA$ *with:*
  - *CSA is the set of all client side artifacts which are HTML Web pages (WP in Figure 4.1) and Client Scripts (CS in Figure 4.1).*
  - *SSA is the set of all server side artifacts which are Server Scripts (SeS in Figure 4.1) and Server Classes (SC in Figure 4.1)*
- *R is the set of all common and possible relationships between artifacts of Web applications with:*
  - *(CSA × SeS) corresponding to relationships between static Web pages and server side scripts (such as* `submit`, `build`, `redirect`, `clientScriptRequest`, *among other relationships in Figure 4.1)*
  - *(SeS × SC) corresponding to relationships between server scripts and server classes (eg.* $(ses_1$ `use` $sc_1)$ *in Figure 4.1)*
  - *(SeS × SeS) corresponding to relationships between server scripts (eg.* `redirect` *in Figure 4.1)*
  - *(SC × SC) corresponding to relationships between server classes (eg.* $(sc_1$ `use` $sc_3)$ *in Figure 4.1)*
  - *(CSA × CSA) corresponding to relationships that exist between client side artifacts (eg.* $(wp_1$ `include` $cs_1)$ *in Figure 4.1)*

For instance, for the example given in Figure 4.1 we have:

- $CSA = \{ wp1, wp2, cs1\}$
- $SSA = \{ ses_1, ses_2, ses_3, sc_1, sc_2, sc_3\}$
- $R = \{(wp_1$ `include` $cs_1)$, $(ses_2$ `build` $wp_2)$,...$\}$

Representation of a Web Interface

A Web (user) interface (UI) may be formally defined as a subgraph of the graph representing the Web application to which it belongs. Thus, a Web UI is defined by a pair (E_{wui}, R_{wui}) such as: $E_{wui} = (CSA_{wui} \cup SSA_{wui})$ with $(CSA_{wui} \subseteq CSA) \wedge (SSA_{wui} \subseteq SSA) \wedge (R_{wui} \subseteq R)$

A Web UI consists of server pages, client static pages and client built pages. The server pages are deployed on the Web server and could manipulate some server classes; client static pages have a static content which is composed of HTML tags; the content of client built pages is generated on the fly by the server pages after processing user's requests.

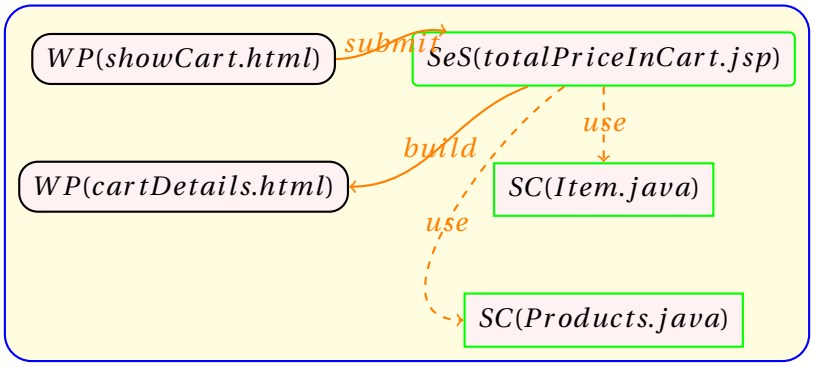

Figure 4.2 : A subgraph representing the Cart Web Interface

Example of a Web Interface SubGraph

Considering our example of the e-shopping application. In the Cart Web UI the total price of all saved items in the user's cart is calculated and presented to users via a Web interface.

Figure 4.2 shows a subgraph that represents the Cart Web UI [1] where,

- $WP = \{ showCart.html, cartDetails.html \}$
- $SeS = \{totalPriceInCart.jsp\}$
- $SC = \{ Item.java, Products.java \}$
- R={($totalPriceInCart.jsp$ use $Item.java$),... }

In this subgraph, the user can access the cart through the client page $showCart.html$. In order to calculate the total price of the products in the cart, the user can submit data (such as product references and quantities) to the server script located in the JSP page $totalPriceInCart.jsp$. As a result, the total price is displayed to the user via the client page $cartDetails.html$.

Properties of server side artifacts

We define a set of structural and behavioral properties related to `server side artifacts` as follows:

- `Request Parameters`: are the data entered by users when manipulating a Web interface and which are processed by a server side script.

[1] This subgraph is used as an illustrative example throughout this thesis

For each element $ses \in SeS$,
$RP(ses)$ is the set of request parameters which are processed by the server script ses.

- `Environment Objects`: Each server side script could manipulate a set of environment objects such as session variables, cookies and business objects in order to store and share user's data.

  For each element $ses \in SeS$,
  $EO(ses)$ is the set of environment objects which are manipulated by the server script ses.

- `Produced Contents`: For each element $ses \in SeS$,
  $PC(ses)$ is the set of produced contents by a server side script as a result of processing user's requests.

- `Statements of Server Scripts`: For each $ses \in SeS$, $Stats(ses)$ is the set of all statements declared in ses.

- `Methods of Server Classes`: For each $sc \in SC$, $M(sc)$ is the set of all methods declared in a server class sc.

- `Method parameters`: Each server method has a set of parameters, where

  For each method $m \in M(sc)$,
  $IPar(m)$ is the set of input parameters of m and $OPar(m)$ is the set of output parameters of m.

- `Navigation Condition`: $NC(wp)$ represents the associated navigation condition to a Web page wp. It states that the user action navigates dynamically from the Web page wp to another page. In most web applications, navigation is not static. The page flow does not just depend on which button the user clicks, but also on the input value that (s)he introduces. For example, submitting a login page may have two outcomes: success or failure. The outcome depends on a computation (result of reference method invocation), namely, whether the username and password are valid.

The presented model and the set of definitions are used later throughout the thesis.

4.3.2 Web Service Oriented System Model

Perepletchikov *et al.* [Perepletchikov *et al.*, 2007] extended the generic model proposed by Briand *et al.* [Briand *et al.*, 1996] and proposed a model covering structural and behavioral properties of the design artifacts in service-oriented systems. We adapt this model for representing the generated Web Service oriented application as a graph. In this graph, the WSDL files are used as service interfaces and object-oriented (OO) classes are implementations for the primitive Web services. BPEL processes are used as implementations of the generated Web

service orchestrations. Fig. 4.3 shows an example of a graph representing the software entities of an imaginary Web service-oriented application.

Definition 2 *Representation of a Web Service-oriented System*

A Web service-oriented system is represented as a pair (E_{sos}, R_{sos})*, where*

- $E_{sos} = SI \cup BP \cup C$;
- SI *is a set of all service (WSDL) interfaces;*
- BP *is a set (possibly empty) of all BPEL processes that implement the WSDL service interfaces of the Web service orchestrations;*
- C *is a set of all OO classes that implement WSDL service interfaces of primitive Web services ;*
- R_{sos} *is the set of all common and possible relationships between the sets* SI, BP *and* C.

  So, $R_{sos} = \texttt{IIR} \cup \texttt{ISR} \cup \texttt{WSR}$ *with,*

  - `IIR` *(Interface Implementation Relationship) represents relationships between service interface and service implementation elements. A Service interface could be implemented using OO classes and/or business processes.*
  - `ISR` *(Internal Service Relationship) represents relationships between classes. Two classes in a given service could have a dependency relationship when an object of the first class invokes the methods (on objects) of the second class.*
  - `WSR` *(Web Service Request Relationship) represents relationships between a class (or a Business process) of a particular service and a service interface of another service. A class (or BPEL process) can invoke the operations defined in the service interface of another service.*

For instance, for the graph given in Figure 4.3 we have:

- $SI = \{si_1, si_2\}$
- $C = \{c_1, c_2, c_3, c_4, c_5, c_6\}$
- $BP = \{bpel_1\}$
- R= {(si_1 `IIR` c_1), (c_1 `ISR` c_2), (c_3 `WSR` si_2),...}

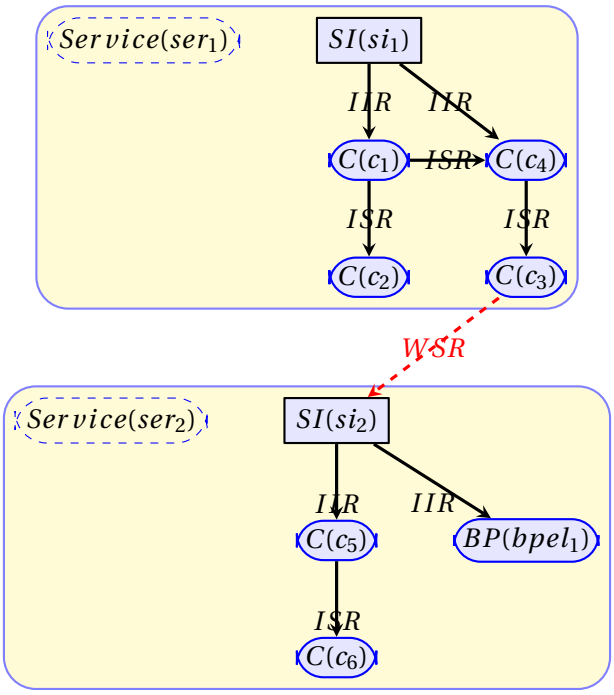

Figure 4.3 : Structure of a Web Service-oriented System

A Web service may be formally defined as a subgraph of the graph representing the Web service-oriented system to which it belongs. Thus, a Web service is defined by a pair (E_s, R_s): $E_s = (SI_s \cup BP_s \cup C_s)$ where $(SI_s \in SI) \wedge (BP_s \subseteq BP) \wedge (C_s \subseteq C) \wedge (R_s \subseteq R)$ with,

- A Web service has only a single service interface SI_s
- Each Web service exposes a set of operations, where
  - For each element $e \in SI \cup BP \cup C$,
    $O(e)$ is the set of operations of e.
  - For each operation $o \in O(e)$, $IParam(o)$ is the set of input parameters of o; $OParam(o)$ is the set of output parameters of o.
  - For each operation $o \in O(e) \wedge e \in C$, $Code(o)$ is the set of all statements of o.

Example of Web service-oriented system graph

Returning to the Cart Web UI presented previously (Section 4.3.1), Fig. 4.4 shows a graph $SOS_{sos1} = (E_{sos1}, R_{sos1})$ that represents the Web services which could be generated starting from this Web UI.

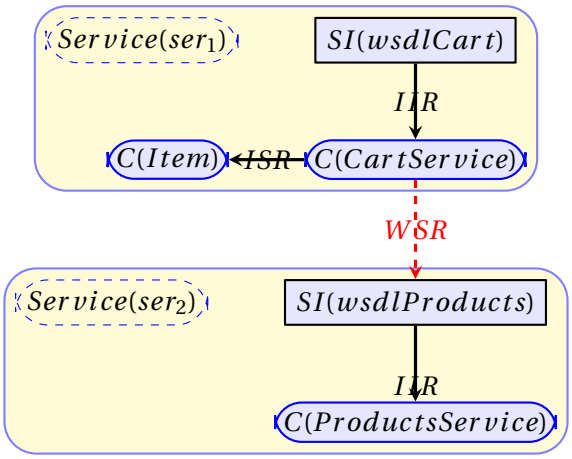

Figure 4.4 : A subgraph that represents the generated Web services from the Cart Web Interface (SOS_1)

- $SI = \{wsdlCart, wsdlProducts\}$
- $C = \{CartService, Item, ProductsService\}$
- $BP = \{\}$
- $R = \{(wsdlCart$ `IIR` $CartService)$, $(CartService$ `WSR` $wsdlProducts)$,...$\}$

In this subgraph, $Service(ser1)$ represents the generated Web service starting from server scripts in the Cart Web UI and $Service(ser2)$ represents the generated Web service starting from the server class $Products.java$. The WSR relationship represents an invocation to an operation published in $wsdlProducts$ interface from the source code of the class $C(CartService)$. This relationship represents an invocation from a server script in the Cart Web UI to a method (this method is exposed later as a Web service operation) located in the $Products.java$ class.

4.4 Summary

In this chapter, we illustrated through a concrete example, the need for shifting starting from Web application systems into Web service-oriented systems. In order to better understand both kinds of software systems and to present more accurately our migration process, we have proposed a model that represents formally these two systems. Thus, we have used this formal model to illustrate how our example of Web application and the desired service oriented system are represented in an unambiguous way as two directed graphs expressed using a set-theoretic notation. The migration process is then considered as a mapping between graphs. The properties that characterize the concepts in both systems are also mathematically defined as sets.

In the two next chapters, we use these formal definitions to present the migration of Web applications toward individual and composite Web services. The approach is explained using a set of procedures and functions that use this formal model.

CHAPTER 5

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

> The rise of Google, the rise of Facebook, the rise of Apple, I think are proof that there is a place for computer science as something that solves problems that people face every day.
>
> Eric SCHMIDT.

5.1 Introduction

This chapter covers the heart of this thesis, we present our solution for migrating Web applications toward Web service-oriented systems. In section 5.2, we give a general overview of the proposed approach. The proposed approach is a multi-step process which is explained through an algorithm proposed in this chapter. All the details about the processing performed on Web applications in order to generate primitive Web services are presented in the remaining sections. In addition, we use our formal models (presented in Chapter 4) to present the migration of Web applications to Web service-oriented systems as a mapping of graphs.

5.2 Approach Overview

The creation of Web services from Web applications is a semi-automatic multi-step process. This is illustrated in Fig. 5.1. The dashed boxes in the process represent steps where the developer is involved. This process begins by receiving from the developer as input the source code and the configuration files of the Web application to be analyzed. A set of primitive and composite Web services are provided as output. To present the details of our approach accurately, we use the previous formal definitions to represent the input Web application as a pair (E, R) and the desired Web service oriented solution as another pair (E_{sos}, R_{sos}). Moreover, we show how to generate the Web service oriented application as a mapping from a Web application graph to a Web service-oriented system graph. The Algorithm 1 introduces a sequence of the main functions and procedures used to apply this mapping. In this algorithm, we follow the same logic of steps in Fig. 5.1. The transformation process is composed of five steps.

A first version of this process was published initially in ECSA conference [Tibermacine et Kerdoudi, 2010], CAL conference [Tibermacine et Kerdoudi, 2011] and ICWS conference [Tibermacine et Kerdoudi, 2012]. After that, our method was detailed and published in the Service Oriented Computing and Applications journal [Kerdoudi *et al.*, 2016].

In the following we give a brief description of each step in the proposed process.

1. **Operation Extraction** First, each element in a Web application is statically parsed to identify the potential set of operations. The operations can be identified starting from existing methods in server classes (see Line 3 in Algorithm 1) and from the server scripts that provide functionalities via Web UIs (see Line 5).

2. **Input and Output Message Identification** The input and output messages related to each identified operation in the Web services are extracted starting from the parsed elements in the Web component. In this step, many aspects of Web application are taken into consideration such as: HTTP requests and HTTP responses, session variables, cookies, client-side scripts and their direct HTTP requests, in-line documents, among others.

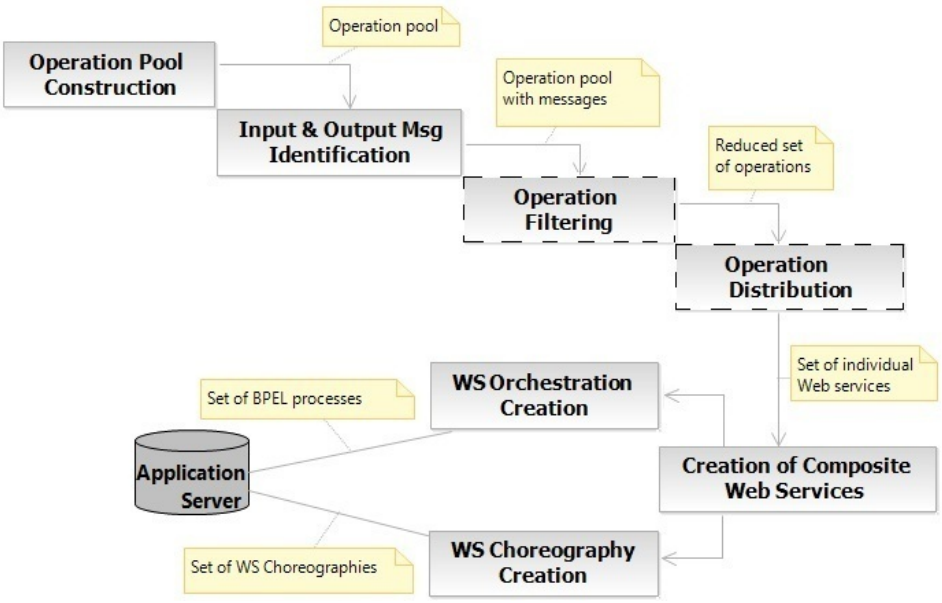

Figure 5.1 : The Proposed Migration Process

3. **Operation Filtering** Then, all operations that must not be published in Web services are eliminated. This needs the contribution of the developer. (S)he should manually remove these unwanted operations. Besides, we provide to the developer a way for specifying constraints so that the operations that are recurrently unwanted can be automatically removed (see Line 9).

4. **Operation Distribution** After that, the remaining operations are grouped in Web services based on two criteria (see Line 10). Similar operations (lexically) are distributed on different Web services to ensure some level of reliability. The tightly 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. Besides, each created Web service is represented as a subgraph of the output graph.

5. **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: (i) Operation-call dependencies which give rise to Web service choreographies. The identified dependencies between Web services are represented using the WSR relationships (see Line 11). (ii) Web navigation relationships which allow to generate a set of Web service orchestrations. The generated orchestrations are represented and added as additional subgraphs of the output graph (see Line 12).

Algorithm 1 From a Web application model to a Web service-oriented system model

Input: Web application model $WA = (E, R)$
Output: Web service-oriented system model $SOS = (E_{sos}, R_{sos})$

```
1: for all e ∈ E do
2:     if e ∈ SC then
3:         s = identifyExistingOperations(e)
4:     else if e ∈ SeS then
5:         s = identifyOperationsFromWebInterfaces(e)
6:     end if
7:     add (s, SOS)
8: end for
9: filterUnwantedOperations(SOS, oclConstraints)
10: distributeOperations(SOS)
11: createChoreographies(SOS)
12: createBPELProcesses(SOS)
13: return SOS
```

Hereinafter, we detail each function in the process. In this chapter, we present the details of the four first steps, which concern the generation of individual Web services. The details of the composition of these Web services is presented in Chapter 6.

5.3 Operation Pool Construction

In this step of our approach a pool of operations is constructed by parsing the Web application's contents. Two kinds of operations are created. This step of our approach has been validated in the publications [Tibermacine et Kerdoudi, 2012] and [Kerdoudi *et al.*, 2016].

5.3.1 Identification of Existing Operations

These operations are generated starting from the existing methods and functions in the back-end components of the Web application. To do so, we statically parse different elements (e.g. classes or server scripts) of the Web components forming the transformed application. All methods in classes and functions in scripts are prepared to be considered as potential operations in Web services. For instance, in the example presented in section 4.3.1 the public methods that are declared in the $Products.java$ class are transformed into new operations. Algorithm 2 shows how to create a service containing operations that are generated starting from existing public methods in a server class.

Therefore, the service interface and its implementation class are created here (see Lines 3 to 5 in Algorithm 2).

Algorithm 3 gives the details of how to create operations. Each access to global variables

Algorithm 2 Identify Existing Operations

```
1: function IDENTIFYEXISTINGOPERATIONS(serverClass : SC)
2:     s = create Service : s=(SI_s, BP_s, C_s, R_s)        ▷ s is subgraph of a service
3:     wsdl = create ServiceInterface : wsdl ∈ SI_s
4:     c = create Class : c ∈ C_s
5:     r = create IIR : r = (wsdl IIR c) ∧ r ∈ R_s
6:     for all m ∈ M(serverClass) do
7:         if isPublic(m) then
8:             createNewOperation(s, c, m, wsdl)
9:         end if
10:    end for
11:    return s
12: end function
```

or attributes from the source code of the identified methods and functions is transformed into additional parameters (see Line 5 in Algorithm 3). This makes these operations stateless. Moreover, each server class and all their dependent internal classes (which does not publish operations) are grouped together to form a Web service (see Lines 7 to 12 in Algorithm 3).

Algorithm 3 Create New Operation

```
1: function CREATENEWOPERATION(s : SOS, c : C_s, m : M(c), wsdl : SI_s)
2:     op = create Operation : op ∈ O(c) ∧ op ∈ O(wsdl)
3:     IParam(op) = IPar(m)
4:     OParam(op) = OPar(m)
5:     IParam(op) = IParam(op) ∪ UsedGlobalVarIn(m)
6:     Code(op) = Stats(m)
7:     for all usedClass ∈ getUsedClassesIn(Code(op)) do
8:         if usedClass does not publish operations then
9:             c_1 = create Class : c ∈ C_s
10:            isr = create ISR : isr = (c ISR c_1) ∧ isr ∈ R_s
11:        end if
12:    end for
13: end function
```

5.3.2 Creation of New Operations from Web Interfaces

The main entities of a Web application are the Web user interfaces that consist of server's pages and client's pages. Through client pages, the end-users can submit data to server-side scripts and through which they can receive the processing results.

The public functionalities which are accessible via Web interfaces from end-users are transformed into new operations. These operations have as parameters the data entered by users when manipulating the Web interface (data entered in forms, for example), and have as output

the results returned by the scripts processing the data entered by the users. In other words, the code present in programs executed at the server-side (JSP or PHP scripts, for example) is grouped within new operations and formatted to be executed as stand-alone code. For example, all the code present in scriptlets of a JSP page which implements a provided functionality to users is grouped and formatted within a single operation. For instance, in the Web interface presented in section 4.3.1, the code present in scriptlets of $totalPriceInCart.jsp$ server page is formatted within a new operation exposed by the $wsdlCart$ interface in Fig. 4.4. More details about this transformation is given in the next section.

Therefore, the Web interfaces that use a secure protocol such as TSL(SSL) [Dierks et Rescorla, 2006] to protect the exchanged messages content or to authenticate the client by using a client's public key certificate are migrated toward a secure Web service that uses the same protocol. Indeed, the source code that implements the TSL(SSL) protocol in the Web interface is formatted to be a secure Web service. In this way, our approach will not weaken the migrated Web application security level.

Besides, a Web interface may include a Frameset composed of one or more frames, and in each frame, there is a content which could be dynamically loaded from elsewhere (Web intefaces, texts, images, etc.). For example, the `<iframe src="URL">` tags are used to embed another content (HTML, JSP, PHP...) within a given HTML, JSP or PHP document. For such tags, the inline frame (document) sometimes corresponds to a Web interface, which has been transformed into a Web service. In this case, an invocation to this Web service is added in the source code of the operation created starting from the currently analyzed Web interface. This invocation allows to retrieve some data from the server.

Algorithm 4 shows how to create a Web service subgraph starting from a Web interface subgraph. First, the service interface and its implementation class are created (see Lines 3 to 5). After that, a new operation is created from the Web interface (see Lines 6 to 9). This operation and its internal dependent classes are grouped within a Web service (see Lines 10 to 15). In other words, for each dependent class, we create a node of type C_s. This node is connected to the class that implements the service with an ISR relationship. For example, in Fig. 4.4 the $C(Item.java)$ class is an internal class used by the $C(CartService)$ class. Additional operations are also generated from the existing methods in the server page (see Lines 16 to 20).

5.4 Input and Output Message Generation

Based on the result of the first step, the input and output messages related to each operation in Web services are identified and generated starting from the parsed elements in the Web application: i) For operations in classes and other structured code elements, the parameters and the returned values are formatted as (respectively, input and output) SOAP messages (see Lines 3 and 4 in Algorithm 3); ii) The saved data in HTTP requests and HTTP responses are parsed

to extract new input and output messages (see Lines 7 and 8 in the Algorithm 4); iii) The used environment objects (session variables, cookies and business objects) by the Web interface are considered as input and output messages (see Lines 7 and 8 in Algorithm 4). This step of our migration process and the used examples have been validated in the publication [Kerdoudi *et al.*, 2016].

Algorithm 4 Identify Operations From Web Interfaces

1: **function** IDENTIFYOPERATIONSFROMWEBINTERFACES($serverClass : SeS$)
2: $s =$ `create` $Service$: s=(SI_s, BP_s, C_s, R_s) ▷ s is subgraph of a service
3: $wsdl =$ `create` $ServiceInterface : wsdl \in SI_s$
4: $c =$ `create` $Class : c \in C_s$
5: $r =$ `create` `IIR` $: r = (wsdl$ `IIR` $c) \wedge r \in R_s$
6: $op =$ `create` $Operation : op \in O(c) \wedge op \in O(wsdl)$
7: $IParam(op) = RP(ses) \cup EO(ses)$
8: $OParam(op) = PC(ses) \cup EO(ses)$
9: $Code(op) = Filter(Stats(m))$
10: **for all** $usedClass \in getUsedClassesIn(Code(op))$ **do**
11: **if** $usedClass$ does not publish operations **then**
12: $c_1 =$ `create` $Class : c \in C_s$
13: $isr =$ `create` `ISR` $: isr = (c$ `ISR` $c_1) \wedge isr \in R_s$
14: **end if**
15: **end for**
16: **for all** $meth \in declaredMethodIn(ses)$ **do**
17: **if** $isPublic(m)$ **then**
18: $createNewOperation(s, c, meth, wsdl)$
19: **end if**
20: **end for**
21: **return** s
22: **end function**

5.4.1 Dealing with HTTP requests and HTTP responses

The code present in the server programs (e.g. server pages like JSP or PHP pages) is parsed to extract the input values received in the HTTP requests (by identifying the statements getting values from HTTP requests). Their types are deduced from the parsed code by analyzing type casts and other conversion statements. This is directly possible in statically typed scripting languages like JSP and C♯. For dynamically typed ones (like PHP or Python), we use external tools for type inference. In addition, the shared objects (such as `JavaBeans` instances), which are used across multiple Web interfaces, are considered as additional input parameters for the generated operation. In this way, the saved data in these objects can be passed from an operation to another in order to compose them.

Furthermore, the contents produced by the server programs, which are viewed at the client side (this content is produced using statements such as: JSP expressions or `out.println(...)` for JSP and PHP's `echo()` or `print()` function calls), are considered as output values. The types of these values are extracted from the code and defined in the generated SOAP messages. The arguments of the `out.println(...)` or `echo(...)` methods can be variables, method invocations or expressions. For variables, we get their type from the parsed code. In the case of method invocations, the type of the generated output message corresponds to the returned type of the invoked method. The expressions can be a concatenation of texts and values of variables and/or method invocations. In this case, the values from method invocations and the variable accesses are extracted to be added as output values of the generated operation. The text is added as another output.

Let us consider the Cart Web UI of our example presented in Section 4.3.1. This interface allows users to calculate the total price of her/his items. Listing 5.1 shows an excerpt of the code present in the $totalPriceInCart.jsp$ server script. Two input messages are identified starting from the `request.getParameterValues(...)` statements (see Lines 8 and 9). They correspond to the `references` of the selected items to be purchased and the `quantities` wanted by the user for each selected item. Another input message is extracted from the used `JavaBean` object `prods`. For the subgraph created for this Web interface (see Fig. 4.2), we represent these values as: RP={$references, quantities$} and EO= {$prods$}.

```
1  <jsp:useBean class="shop.prod.Products" id="prods" scope="page"/>
2  <%!
3  String[] references;
4  String[] quantities;
5  double totalPrice = 0;
6  %>
7  <%
8  references = request.getParameterValues("references");
9  quantities = request.getParameterValues("quantities");
10 if(references != null){
11 for(int i = 0; i < references.length; i++){
12 Item item = prods.getItemByReference(references[i]);
13 double unitPrice = item.getUnitPrice();
14 int quantity = Integer.parseInt(quantities[i]);
15 totalPrice = totalPrice + calculateTPrice(unitPrice, quantity);
16 }
17 }
18 %>
19 <%= totalPrice %>
```

LISTING 5.1 : An excerpt of the code present in the `Cart` Web interface

The type of the `references` input is an array of Strings. From the conversion statement (see Line 14) we have deduced that the type of `quantities` is an array of Integers.

Finally, we have deleted from the source code of the generated operation: the conversion, the cast and the `request.getParameterValues(...)` statements (Lines 8, 9 and 14 in Listing 5.1). The returned value of the generated operation is the result saved in the variable `totalPrice` and bound to the Web interface using a JSP expression (Line 19). This is used to create an output SOAP message of type `Double` (in our subgraph, we have $PC = \{totalPrice\}$). After applying the two first steps and making the necessary modifications for the previous source code, a new operation is created (see Listing 5.2). For that, four input parameters and a returned value are identified.

```
1  public double serviceTotalPriceInCart(String[] references, int[] quantities, shop.
       prod.Products prods){
2  double totalPrice = 0;
3  if(references != null){
4  for(int i = 0; i < references.length; i++){
5  Item item = prods.getItemByReference(references[i]);
6  double unitPrice = item.getUnitPrice();
7  int quantity = quantities[i];
8  totalPrice = totalPrice + calculateTPrice(unitPrice, quantity);
9  }
10 }
11 return totalPrice;
12 }
```

LISTING 5.2 : An excerpt of the generated operation from the `Cart` Web interface

Fig. 4.4 shows the subgraph that represents the generated Web service from the `Cart` Web interface and the server class `Products`, where, the following values represent these services.

- $O(CartService) = \{totalPriceInCart, calculateTPrice, ...\}$
- $IParam(totalPriceInCart) = \{references, quantities, prods\}$
- $OParam(totalPriceInCart) = \{totalPrice\}$
- $IParam(calculateTPrice) = \{unitPrice, quantity\}$
- $OParam(calculateTPrice) = \{TPrice\}$
- $O(Products) = \{getItemByReference, getItemDetails, ...\}$

5.4.2 Handling Session Objects

Most of Web applications manage session variables in order to store and share the user's data when (s)he navigates from a Web interface to another one. To avoid losing this data and make possible using them in the composition of the generated Web services from the Web interfaces, we consider these values as additional input and output messages.

The user's data that is stored in these session objects could be used as constraints for accessing other Web interfaces when the user navigates in the application. Therefore, the output messages that are generated from the first Web interface are considered as the input to the generated services from the navigated Web interfaces. This ensures that these services are not freely accessible (preserve security). In order to generate these messages, we parse the code present in operations that use session variables.

In our illustrative example, the `addItem` Web interface is used to add a new item into a virtual cart. Listing 5.3 shows an example of using session variables to store information about the cart. The `quantity` and the `reference` are identified and considered as input messages. They are identified after the parsing of the `request.getParameter(...)` statements (as explained in the previous section).

```
<%
Cart currentCart =(Cart) session.getAttribute("currentCart");
// return the object bound with the name 'currentCart' in this session, or null if
    no object is bound under this name
Cart newCart = null;
if (currentCart == null){
//binds a new object 'newCart' to this session using the name "currentCart"
newCart = new Cart();
session.setAttribute("currentCart",newCart);
}
else {
newCart = (Cart) session.getAttribute("currentCart");
}
String StrQuantity = request.getParameter("quantity");
int quantity = Integer.parseInt(StrQuantity);
if (quantity > 0){
String reference = request.getParameter("reference");
// update the Cart and the object bound the this session
newCart.addItem(reference,quantity);
}
out.println(newCart.getTotalPrice())
%>
```

LISTING 5.3 : An excerpt of a server script present in the `addItem` Web interface

Now, statements such as `session.getAttribute(...)` (see Lines 2 and 11) return the objects bound to the name `'currentCart'` specified in this session. So, this object is transformed into an additional input. The session objects can be updated in the source code (see Line 18). For this reason, they are also considered as output messages of the generated operation. After analyzing the cast statements in Lines 2 and 11, we have deduced that the concrete type of the `'currentCart'` message is `Cart`.

In addition, the parsing of statements that are used to bind an object to a session such as: `session.setAttribute(...,...)` generates additional output messages. For example, from the Line 8, we create an output message named `'currentCart'`. At the end, an additional output message corresponding to the calculated price is generated from the parsing of the statement `out.println(...)` (see Line 20). The type of this output corresponds to the returned type of `getTotalPrice` method that is `Double`. Listing 5.4 shows an excerpt of the newly created operation. It allows to add a new item in the virtual cart. This operation receives three input messages (`reference`, `quantity` and `currentCart`) and returns a composed message that contains the new total price and the updated virtual cart. However, the statements that use session variables are removed from the code of this operation.

```
1  public AddItemOutput serviceAddItem(String reference, int quantity , Cart
       currentCart ){
2  Cart newCart = null;
3  if (currentCart == null){
4  newCart = new Cart();
5  currentCart = newCart;
6  }
7  else {
8  newCart = currentCart;
9  }
10 if (quantity >0){
11 newCart.addItem(reference,quantity);
12 }
13 return (new AddItemOutput(newCart.getTotalPrice(), currentCart));
14 }
```

LISTING 5.4 : An excerpt of the generated operation from `addItem` Web interface

5.4.3 Dealing with Cookies

Actually, the server can maintain information about user sessions in many ways such as using cookies. In our approach, the used set of cookies is considered as input and output messages. The statements which are used to access and to modify the saved cookies (e.g. in a JSP page, `request.getCookies()` and `response.addCookie(...)`) are identified and replaced in the body of the new generated operation with equivalent statements accessing these new mes-

sages. Listing 5.5 shows an example of using cookies to save information about user authentication in the `signIn` Web interface.

The parsing of this code has identified one input message which is defined starting from the `request.getCookies()` statement (see Line 2). This input value corresponds to a set of cookies that are used in the generated operation's code. This set of cookies is returned as output message of this operation in order to be used as input of another operation generated from another Web interface that uses these cookies. In this way the composition of these two operations would be easier.

```
1  <%
2  Cookie[] cookies = request.getCookies();
3  String email = "", password ="";
4  if( cookies != null ){
5  for(Cookie cookie : cookies){
6  if (cookie.getName().equals("email")){
7  email = cookie.getValue();
8  }
9  if(cookie.getName().equals("password")){
10 password = cookie.getValue();
11 }
12 }
13 AccountManager userId= new AccountManager();
14 if(userId.signIn(email,password)){
15 // ....
16 }
17 }
18 %>
```

LISTING 5.5 : An excerpt of code showing the using of cookies in the `signIn` Web interface

The generated operation has a body with the same code as the script shown above, except the statement at Line 2. This Line is replaced with a statement which is used for extracting the cookies from an object of type Collection (`obj.getCookies()`) received as an argument. In addition, a "`return`" statement is added at the end of the operation's body, which returns this object (`retun obj;`).

5.5 Operation Filtering

In this step, the identified pool of operations is filtered by eliminating the operations which are not suitable to be published in Web services. For example, the modern Web applications use Public and Private APIs. Thus, by this filtering task, we allow developers to eliminate all operations that are identified from Private APIs. The filtering cannot be fully automated and it

needs the developer involvement. The developer is asked to choose among the selected operations those that are not interesting for a publication. A set of filtering expressions are made available to be used and enriched by the developer. Some kinds of operations are recurrent in most of applications. Therefore, the specified expressions could be reused by another developer in order to filter operations which are generated starting from other Web applications. The developer will not have to specify them from scratch. These expressions are constraints that are checked on an Ecore [Eclipse, 2009a] instance of a meta-model representing operations. These instances of the meta-model are automatically built by analyzing the operations' code. Constraints are Boolean expressions which are specified using OCL (Object Constraint Language [OMG., 2006]). OCL has been chosen because of its simplicity [Briand *et al.*, 2005] and the existence of a good tool support (OCL Toolkit [Dresden., 2009], Eclipse MDT/OCL [Eclipse, 2009b], ...). The specified constraints navigate in the meta-model, which is illustrated in Figure 5.2. This meta-model is an excerpt of the UML meta-model (related to operations) [OMG., 2011b] extended with some basic constructs.

The main meta-class in Figure 5.2 is `Operation`, which represents an identified operation from the code. The `Operation` meta-class is associated to a `Type` meta-class which represents the returned type of the operation. In addition, this operation could have a body and a set of parameters. All constraints have as a context an instance of the `Operation` meta-class. An example of a constraint is given below:

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

This OCL constraint states that the operations which use the session standard script variable must not be selected.

```
context Operation inv :
not ((self.returnedType.name= 'void') and
(self.name.substring(1,3) = 'set') and
(self.ownedParameter->size() = 1) and
self.body.statement->exists( kind = 'AssignmentStatement' and
isFieldAccess = 'true')
```

In this example, all operations that represent field accessors (for example, setter methods) are eliminated.

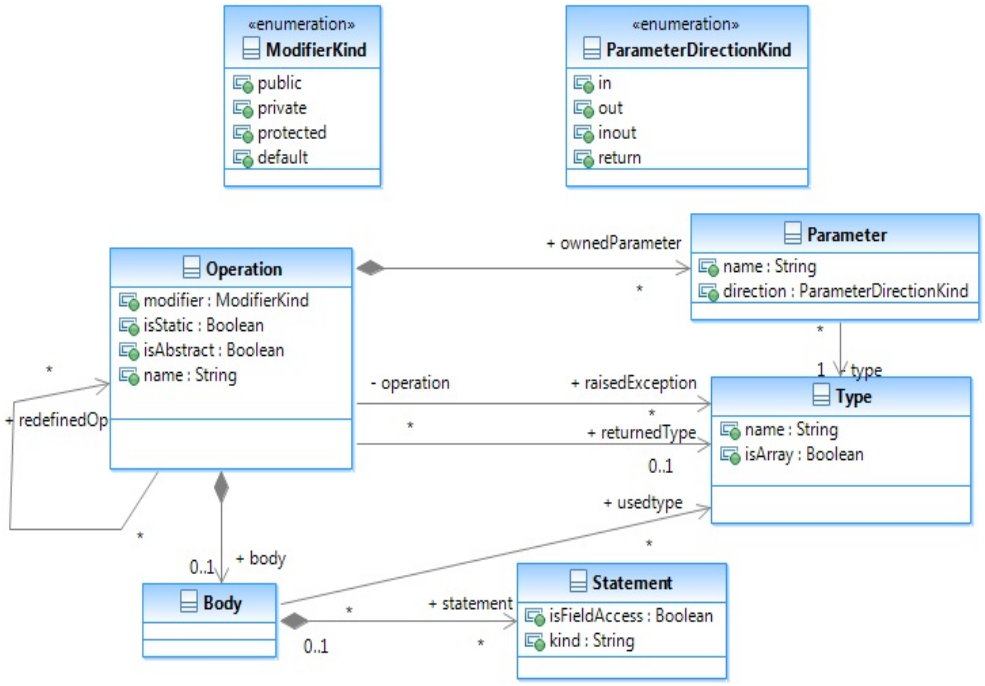

Figure 5.2 : The Operation Meta-model

The operation filtering approach has been validated in the publications [Tibermacine et Kerdoudi, 2010], [Tibermacine et Kerdoudi, 2012] and [Kerdoudi *et al.*, 2016].

5.6 Operation Distribution in Services

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

5.6.1 Grouping Criterion

We group operations based on the cohesion and coupling criteria. We argue that an optimal granularity is the key to a well-designed service. Service granularity generally refers to the performance and size of a service [Kulkarni et Dwivedi, 2008]. The data granularity is one of service granularity types [Haesen *et al.*, 2008]. It reflects the amount of data that is exchanged with a service. A good grouping of the identified operations in Web services has a positive impact on the data granularity. In our approach, the highly coupled and cohesive operations are grouped together in a single Web service. And, the loosely coupled operations are distributed on multiple services. This strategy of grouping reflects a low amount of data which is exchanged and reduces the communication overhead. Hence from the service quality viewpoint, we increase the performance and the maintainability of the generated services.

The cohesion of a service is assessed based on the degree of the strength of functional relatedness of operations within a service. We measure the cohesion of a service by analyzing the static invocations between operations within that service. Several cohesion metrics have been proposed in the literature in order to measure the cohesion of a class in an object-oriented system [Briand *et al.*, 1998]. We believe that one of these metrics can be used to evaluate the cohesiveness of the generated Web services. The *LCOM* (Lack of Cohesion in Methods), *TCC* (Tight Class Cohesion) and *LCC* (Loose Class Cohesion) are one of the most used metrics to measure cohesion between public methods in a class. The problem with *LCOM* metric is that such metric only helps in identifying the absence of cohesion rather than its presence [Etzkorn *et al.*, 2004]. On the contrary, we need in our work to measure the presence of cohesion in Web services. For this reason we use the *TCC* and *LCC* metrics to measure service cohesion. To do so, we start with a flat organization of operations (all operations are distributed in one Web service). Then, we calculate *TCC* and *LCC* to check the cohesiveness of this Web service. If the service is not cohesive, we split it into set of low coupling services and we check again the cohesiveness of each one of them. We repeat the measurement until the produced services are “quite cohesive".

To measure the *TCC*, we consider a Web service with N operations. NP is the maximum number of operation's pairs:$NP = [N * (N-1)]/2$. The *NDC* is the number of direct connections between operations. Then *TCC* is defined as the relative number of directly connected operations: $TCC = NDC/NP$.

To measure *LCC*, we consider NIC is the number of indirect connections between operations (when two operations are connected via other operations). *LCC* is defined as the relative number of directly or indirectly connected operations: $LCC = NDC + NIC/NP$.

According to [Badri et Badri, 2004], a class (a service in our case) is considered non-cohesive when $TCC < 0.5$ and $LCC < 0.5$. If $LCC = 0.8$ the class is considered "strongly cohesive". If $TCC = LCC = 1$ then the class is maximally cohesive, which means all methods are connected. In our approach, we experimentally tested these metrics and we found out that with $TCC \geqslant 0.5$ or $LCC \geqslant 0.5$ the obtained Web services are “quite cohesive”.

Algorithm 5 shows how the grouping is performed. First, each group of operations is represented with a graph which is expressed as a pair (OP, CON), where *OP* symbolizes a set of operations. *CON* is a binary relation on $CON(\subseteq OP \times OP)$. It corresponds to the direct and indirect connections between operations.

The input of this algorithm is a group that contains all the identified operations. The output is a set of cohesive Web services.

First, we try to find the best grouping (best level of cohesiveness) by measuring the *TCC*. If $TCC \geqslant 0.5$ we conclude that the Web service is "quite cohesive". In this case, we do not need

Algorithm 5 Grouping Operations

```
1: function GOUPINGOPERATIONS(CL = (OP, CON)))
2:     TCC = calculateTCC(CL)
3:     if TCC ⩾ 0.5 then
4:         return TRUE
5:     else
6:         LCC = calculateLCC(CL)
7:         if LCC ⩾ 0.5 then
8:             return TRUE
9:         else
10:            if existExplicitGroups(CL) then
11:                explicitGroups = split(CL)
12:                for all group ∈ explicitGroups do
13:                    goupingOperations(group)
14:                end for
15:            else
16:                implicitGroups = getImplicitGroups(CL)
17:                for all group ∈ implicitGroups do
18:                    goupingOperations(group)
19:                end for
20:            end if
21:            return FALSE
22:        end if
23:    end if
24: end function
```

to calculate the LCC, because, the existing number of direct connections is enough, in order to know if service is cohesive or not. Now, in the case of $TCC < 0.5$, the indirect connections between operations that belongs to a service are used to assess the cohesiveness of that service. Hence, we need to calculate the LCC. If $LCC \geqslant 0.5$ we consider the Web service is "cohesive" although the $TCC < 0.5$. Now, if the $LCC < 0.5$ and $TCC < 0.5$, then the service is not cohesive. In this case, we split the service into a set of explicit groups of operations (where, there are no connections between these groups) (see Line 11). For each explicit group we invoke again the grouping algorithm. Now, if there are no explicit groups, we identify the implicit groups where there is a lowest coupling between them (see Line 16). And, we repeat the measurement for each group.

For instance, the following operations are created starting from the e-shopping application:

- (op_1) : boolean serviceLogin(String userName, String paswword)
- (op_2) : boolean authenticate(String userName, String paswword)

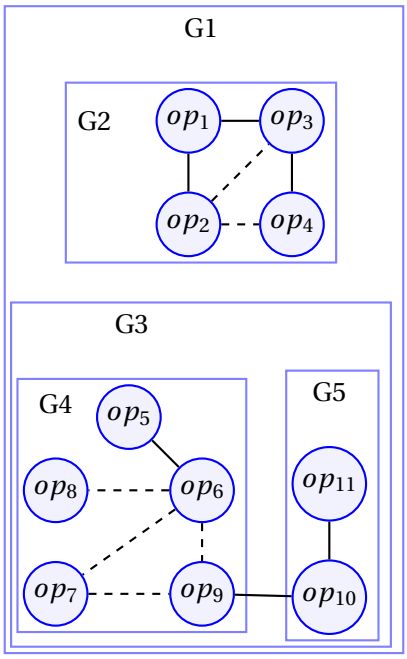

Figure 5.3 : Example of operations grouping

- (op_3) : User getUserDetails(String userName)
- (op_4) : boolean userRegistration(String firstname, String lastname, String address, int mobile, String email, String password)
- (op_5) : Item[] serviceProducts()
- (op_6) : Item[] getAllItems()
- (op_7) : Item[] getItemsByUser(String userName)
- (op_8) : Item getItemDetails(String reference)
- (op_9) : Item getItemByReference(String reference)
- (op_{10}) : Double serviceTotalPriceCart(String[] references, int[] quantities)
- (op_{11}) : Double calculateTPrice(Double unitPrice, int quantity)

Fig. 5.3 shows the dependencies between operations where, dashed lines represent the indirect connections between a pair of operations and solid lines represent direct connections between them. The measurements give the following values:

- TCC(G1) = $\frac{6}{55}$ = 0.10
- LCC(G1) = $\frac{6+6}{55}$ = 0.21
- TCC(G2) = $\frac{3}{6}$ = 0.5
- LCC(G2) = $\frac{3+2}{6}$ = 0.83
- TCC(G3) = $\frac{3}{21}$ = 0.14
- LCC(G3) = $\frac{3+4}{21}$ = 0.33
- TCC(G4) = $\frac{1}{10}$ = 0.10
- LCC(G4) = $\frac{1+4}{10}$ = 0.5
- TCC(G5) = $\frac{1}{1}$ =1
- LCC(G5) = $\frac{1}{1}$ = 1

We have started with the group G1 which is not cohesive. G1 is divided into two explicit groups (G2) and (G3). After that, (G4) and (G5) are created starting from (G3). Finally, the obtained cohesive Web services are: G2 (op1, op2, op3, op4), G4(op5, op6, op7, op8, op9) and G5(op10, op11).

5.6.2 Spreading Criterion

Similar operations are spread out in different Web services. In this way, for users of an operation within a service, another service containing a similar operation can be easily and quickly found at the same provider (reliability).

In other words, Web services are exposed to errors and failures for many reasons, such as, the network is unreachable, the application server is unavailable or the service is not working properly. Hence, the reliability of the programs (it could be an orchestration of Web services) that invokes these services will be decreased. Several error-handling approaches are proposed in the literature such as [Aït-Bachir, 2008], [Crasso *et al.*, 2008], [Kokash, 2006] and [Tibermacine *et al.*, 2015]. Many of these approaches are based on finding a relevant service substitute that replaces the failed service. For example in [Azmeh *et al.*, 2011] and [Tibermacine *et al.*, 2015] the identification of the substitute is based on the measurement of similarity between service interfaces.

In this work, a solution based on a comparison of operation signatures has been used. The WSSim tool [Tibermacine *et al.*, 2013] allows to measure the similarity between operations by comparing the operations' names and input and output messages. Table 5.1 shows the similarity measurement results that are produced by WSSim for the operations that are depicted in

Table 5.1 : Obtained similarity scores

	op	op_2	op_3	op_4	op_5	op_6	op_7	op_8	op_9	op_{10}	op_{11}
op_1	1	0.74	0.53	0.73	0.57	0.41	0.51	0.50	0.49	0.63	0.55
op_2		1	0.52	0.5	0.45	0.42	0.67	0.60	0.55	0.52	0.61
op_3			1	0.53	0.55	0.61	0.70	0.70	0.56	0.38	0.50
op_4				1	0.46	0.35	0.53	0.56	0.53	0.59	0.53
op_5					1	0.42	0.49	0.50	0.49	0.52	0.47
op_6						1	0.63	0.64	0.60	0.34	0.49
op_7							1	0.76	0.79	0.38	0.51
op_8								1	0.86	0.48	0.56
op_9									1	0.59	0.53
op_{10}										1	0.65
op_{11}											1

Fig. 5.3. In this table, we give a score of similarity (between 0 and 1) for all pairs of operations. The operations that have a similarity score that ranges between 0.80 and 1 are considered highly similar. According to the obtained similarity assessment, we consider that operations op_8 and op_9 are highly similar.

After the calculation of cohesion and similarity between the different operations, the next step is to distribute these operations on Web services. In the current implementation, we assist the developer for giving a new organization based on the obtained results from the cohesion and similarity values. The proposed organization could then be manually updated by the developer. For example, we decide to move the operation (op_8) from the Web service G4 into G5. This moving does not have a negative impact on the cohesiveness of the obtained services. This technique of operations distribution based on the grouping and spreading criteria has been validated in the publication [Kerdoudi *et al.*, 2016].

5.7 Web Service Deployment

The validated set of Web services is 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 tool we developed if the hosting server is running on the same machine and if the developer/administrator has all the access rights to this server. He should only specify the file system path to the public directory of the server, which contains the Web service implementations.

Another alternative to the centralized deployment is the remote hosting. In this case, the developer has thus the opportunity to host her/his Web services in a server which is running in another execution context. The code is uploaded and then put in the correct directory of the Web server. This is done via FTP and requires that the developer has all access rights. This

step of our approach has been validated in the publication [Tibermacine et Kerdoudi, 2010], [Tibermacine et Kerdoudi, 2011] and [Tibermacine et Kerdoudi, 2012].

5.8 Summary

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 [Finkelstein et Kramer, 2000]. In this chapter, we presented our solution of migrating Web component based application to Web Services: Towards Considering a "Web Interface as a Service". The proposed solution, is a multi-step process for the transformation of Web 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 potential operations to be published into Web services. All operations that must not be published in Web services are eliminated. We provide a semi-automatic way of eliminating unwanted recurrent operations. This is done through OCL constraints that developers should specify on a simple metamodel of operations inspired from the UML metamodel. These constraints are automatically checked. The operations kept must satisfy these constraints, if any. After that, in order to create Web services with a good level of granularity, we proposed a technique for distributing operations into Web services. Operations tightly coupled are grouped in the same Web services in order to enhance their performance, and operations that are similar are spread-out in different Web services, in order to enhance their reliability. All of the resulting services are deployed on the Web server in order to allow third party development. In our work, we considered 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.

In the next chapter, we detail the step of the Web service composition. We show how the individual Web services are composed automatically based on existing dependences between Web components.

CHAPTER 6

Generation of composite Web Services

Simplicity is prerequisite for reliability.
Edsger DIJKSTRA

6.1 Introduction

In Chapter 5, we presented all the details of our method for transforming component-based Web applications into service-oriented ones. We have illustrated how we assist the developer so that (s)he creates primitive Web services starting from Web applications. In this chapter, we show how to create composite Web services by assembling automatically the generated individual Web services. The potential dependencies between the different selected operations in the Web services are identified. We distinguish two kinds of dependencies between operations: operation invocation dependencies and Web navigation relationships. In Section 6.2, we present the details of the first kind of dependencies, where choreographies are generated by analyzing the relationships between the published Web services. In Section 6.4, we present the details of the second kind, where, the navigations between Web interfaces of the Web application enable to create orchestrations of the Web services that have been created from these Web interfaces.

6.2 Web Service Choreography Creation

The Web service choreographies are created starting from the analysis of the source code of the generated Web services. We identify in the source code of the generated Web services all external calls between operations in order to replace them by Web service requests (WSR). This is illustrated and explained in Algorithm 6. Indeed, 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 (see Line 6). If the called operations are present in the other published Web services these operation dependencies are replaced by Web service requests in source code of the invoking operation. In the created graph for the Web service-oriented system, we represent each Web service request by a relationship of type WSR (a Web Service Request is an invocation of a service from the code of a service client (it could be another Web service)) which relates the invoking class node and the WSDL interface node of the invoked service (see Lines 6 to 9). An example of this relationship is given in Fig. 4.4, where the invocation to the method $getItemByReference$ from the code of the operation $serviceTotalPriceInCart$ is transformed into WSR between the class $CartService$ and the interface $wsdlProducts$. As for a local method invocation, this is represented by an ISR relationship. Fig. 4.4 shows an ISR relationship between the $CartService$ class and the $Item$ class.

Besides, other Web service requests can be also created starting from the parsing of client-side scripts. Indeed, the majority of modern Web applications use Ajax, which allows to build dynamic and interactive Web applications. Client-side scripts can create direct connections to the server and transfer data from clients to servers. The `XMLHttpRequest` API is the mostly used technique as an Ajax implementation [Flanagan, 2011]. In some cases, the request sent to the server asks for a server-side program which has been transformed into a Web service. We

Algorithm 6 Web Service Choreography Creation

1: **procedure** CREATECHOREOGRAPHIES(SOS)
2: **for all** $op \in SOS$ **do**
3: **for all** $invOp \in invokedOpsFrom(op)$ **do**
4: $c_1 = declaringClass(invOp)$
5: $c = declaringClass(op)$
6: **if** $c_1 \notin C_s$ **then**
7: $wsdl_1 = getServiceInterfaceOf(c_1)$
8: $wsr =$ `create WSR :` $wsr = (c$ `WSR` $wsdl_1) \wedge wsr \in R_{sos}$
9: $createWSRequest(Code(op), invOp, wsdl_1)$
10: **end if**
11: **end for**
12: **end for**
13: **end procedure**

check this by the parsing of the scripts that use this API. In this case, we create a new request to this Web service. This request is added at the beginning of the source code of the Web service generated starting from the current Web interface. This invocation allows to update the data at server side before it will be used by the Web service.

6.3 Example of Choreography Creation at Code Level

Let us consider our illustrative example to show how to create a composite Web service at code level. In the `Cart` Web Interface, we use a client-side script to send data to server. Before proceeding to checkout, the user can modify the quantity of the purchased items. The new quantity is sent as data to the server via a client-side script (in JavaScript using an `XMLHttpRequest` object).

This client side script contains a call to a program executed at the server side corresponding to the `updateCart` Web interface. This program allows to update the cart and calculate the new total price. When the user clicks on the `proceed to checkout` button, a program (provided by the `Payment` Web interface) is executed at server side. This program takes as input the new calculated total price and the new cart details.

Following our approach, the `payment` and `updateCart` operations are created respectively starting from the `Payment` and `updateCart` Web interfaces. So, an invocation to `updateCart` operation is added at the beginning of the `payment` source code. In this way, the new total price is calculated before proceeding to payment.

In addition to the invocation of the `updateCart` operation several other operation invocations are created in order to accomplish the payment process. Indeed, the opera-

tions `checkCreditCard` and `checkPersonalInformation` operations are invoked successively. These operations are used respectively, to check the validity of the input credit card information and the personal information of the user. If they are valid, we call two other operations, which are `approvalPayment` operation to approve the payment and `sendEmail` operation. The `approvalPayment` operation itself calls some other operations, which are respectively: `checkCredit`, `createInvoice`, `validatePayment` and `getDeliverySchedule`.

The choreography creation approach has been validated in the publication [Tibermacine et Kerdoudi, 2012] and [Kerdoudi *et al.*, 2016].

6.4 Web Service Orchestration Creation

In this step of our approach, a set of Web service orchestrations is generated from the relationships between Web interfaces. We parse the navigation documents such as JSF `faces-config` files and their navigation rules. This allows the identification of other potential collaborations of the different Web services created from these pages.

6.4.1 Navigation Rule Extraction

In some Web applications the navigation rules are not available. In this case, the hypertext links and the redirection statements/tags located in the Web application are parsed in order to create these rules. The Web application graph could be considered as a navigation model for the input Web application. We associate to each Web page node a navigation condition (NC(web page)). Indeed, the redirection statements like `response.sendRedirect("url")` are generally declared in the body of a conditional statement such as `If Statement`. Thus, depending on the condition value, the page is redirected to the appropriate destination. The used condition in this code is extracted and analyzed to be added as a condition of created navigation rule. This task requires sometimes the developer intervention to validate the generated navigation rules. The process of the extraction of navigation rules has been validated in the publication [Kerdoudi *et al.*, 2016].

6.4.2 BPEL Process Creation Algorithm

The generated BPEL processes represent new services which implement some coarse grained functionalities provided by the application. The exchange of messages between the BPEL process and external clients (other applications or partner (Web) services) is done via a contract described in WSDL. This contract represents an interface of the BPEL composite Web service. Now, the generation of the Web service orchestration is implemented according to Algorithm 7.

In this algorithm, first all navigation paths are calculated from the Web navigation document of the parsed Web application (Line 2). Each path represents a coarse grained func-

Algorithm 7 WS Orchestration Creation Algorithm

```
1: procedure CREATEBPELPROCESSES(naviRules)
2:     naviPaths = calcNaviPaths(naviRules)
3:     for all path ∈ naviPaths do
4:         process = ProcessFactory.newInstance()
5:         seq = process.createSequence()
6:         returnedVal1 = process.createVariable()
7:         for all navigRule ∈ path do
8:             opFrom = parseSourceView(navigRule.sView)
9:             if !(opFrom isPreviouslyInvokedIn process) then
10:                op1 = process.createInvocationTo(opFrom)
11:                op1.setParameters(variablesofprocess)
12:                returnedVal1 = op1.invoke()
13:                process.store(opFrom, returnedVal1)
14:                seq.add(op1)
15:            else
16:                returnedVal1 = getStoredReturnedValBy (opFrom)
17:            end if
18:            ifActivity = parseConditionExpression(navigRule, process, seq)
19:            opTo = parseDestinationView(navigRule.dView)
20:            if !(opTo isPreviouslyInvokedIn process) then
21:                op2 = process.createInvocationTo(opTo)
22:                matchedParts = calculateSimilarity(opFrom.outputMsg, opTo.InputMsg)
23:                op2.setParameters(variablesinprocess+returnedVal1, matchedParts)
24:                returnedVal2 = op2.invoke()
25:                process.store(opTo, returnedVal2)
26:                ifActivity.add(opTo)
27:            else
28:                seq = addPickToProcess(navigRule, process, seq)
29:            end if
30:        end for
31:    end for
32: end procedure
```

tionality provided to the user when (s)he navigates between the Web interfaces of this path. Therefore, for each path we create a BPEL process (Line 4) which represents a new generated composite Web service. After that, for each navigation rule in the current path, we identify the source operation (Line 8). The source operation corresponds to the operation that has been generated starting from the navigation's source Web interface. The same thing is done for the destination Web interface (Line 19). As specified in the algorithm, a navigation rule contains three elements: i) a source view (Line 8), which represents the Web interface(s) from which the navigation started (e.g., the Web interface presenting the form for searching items in the example introduced previously: `search.jsp`); ii) a destination view (Line 19) that corresponds to the Web interface(s) to which the user will be automatically directed (e.g., the Web interface(s) presenting the result of the search `searchResult.jsp`); and iii) an execution condition which contains an expression and a value (e.g., the expression is a call to the JavaBean method for getting the number of items found: `#{searchResult.getItemsCount}`, and the value is `"NotZero"`).

For each navigation rule, we first test if the source operation has already been called in the process while parsing another navigation rule (Line 9). 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 16). This kind of values is stored after each operation invocation (see Lines 13 and 25). Then, we parse the condition part in the navigation rule, by calling a function (see Line 18). In this function, we get the expression of the condition, which corresponds to the operation to be invoked. We test if this operation is not invoked previously in the process. In this case, we create in the process an invocation to this operation. After that, we create in the process an `If Activity`. It is used to compare the obtained value after invoking the operation defined in the condition's expression and the condition's value. If the two values are equal, then we invoke the corresponding destination operation (see Lines 19 to 24). We need to test before, if the destination operation has already been called in the process while parsing another navigation rule. This means that there is a cycle in this navigation path. The cycles occur when the user wants to navigate again in a path with new input values. So, all invoked operations in this cycle can be invoked again with the new input values. We deal with cycles by calling of the function `addPickToProcess` (see Line 28). Fig. 6.1 shows an excerpt of an abstract process, which represents the sequence of generated activities to be added in the process after calling the `addPickToProcess` function. This abstract process contains mainly the activities: `Pick` and `RepeatUntil`(Loop).

The loop activity is used to repeat the set of created invocations starting from the cycle. The pick activity allows the process to block and wait for one or a set of suitable message(s). The arrival of a message indicates that the user needs to repeat the invocation of operations in the cycle. When one of these messages is received, the associated activity is performed and the pick completes. If none of these expected messages is received within a certain period

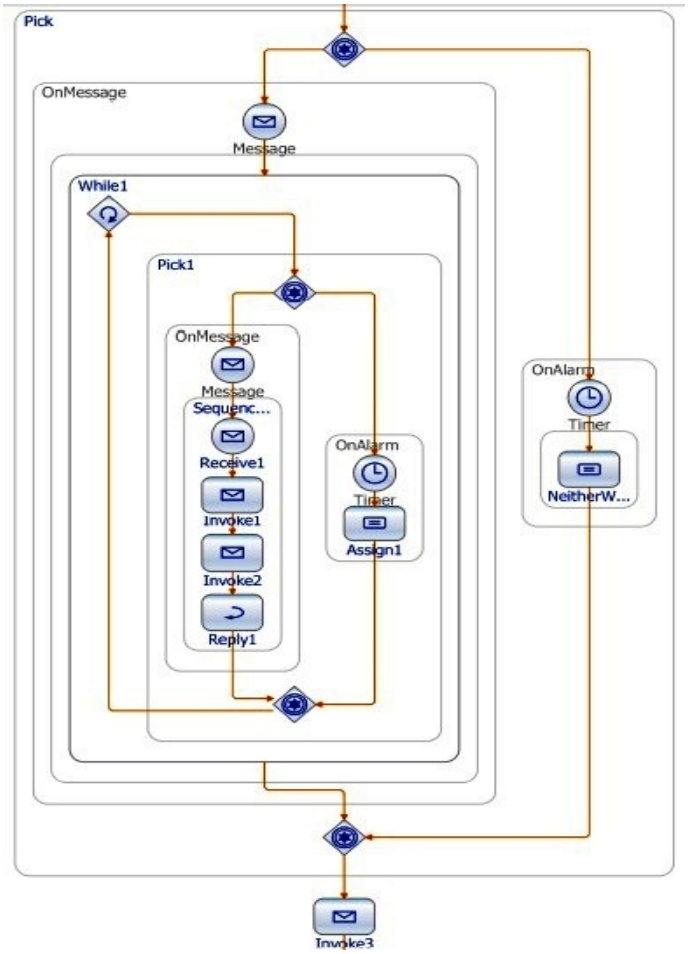

Figure 6.1 : An excerpt of a BPEL process representing the created activities to deal with a cycle.

of time [1], the pick can specify an exceptional behavior to be performed (in our algorithm, an `OnAlarm activity` is added, which allows to the process to wait). In this way, the cycle of the navigation path is considered in the BPEL process.

In orchestration creation, before each operation invocation (see Line (24) in the algorithm above), we prepare the list of arguments. A matching of the variables' names in the orchestration and the arguments of the operation to be invoked is performed. In this way, we check the syntactic composability [Medjahed et Bouguettaya, 2005] of the two operations to be composed and we ensure that arguments are passed in the correct order (see Lines (22) and (23)).

Each generated composite service is modeled as a subgraph that belongs to the graph

[1] We give a default value for this time interval, which could be modified by the developer on the generated BPEL process

which represents the migrated service-oriented system. In this subgraph we create a WSDL interface node and BP(bpel) node and we connect them via an IIR relationship. These nodes represent, respectively, the interface and implementation of the generated composite service. The invocations to operations which belong to other Web services are represented with WSR relationships. These relationships relate the BP(bpel) node of the generated composite service and the WSDL interface node of each invoked service.

The first version of the proposed algorithm has been published in [Tibermacine et Kerdoudi, 2011] and [Tibermacine et Kerdoudi, 2012]. The proposed abstract model which is used to deal with cycles is validated in the publication [Kerdoudi *et al.*, 2016].

6.5 Example of BPEL Process generation

Let us take the example presented in Section 4.2. A BPEL process is generated for the navigation path which represents a successful purchasing. The process first invokes the `basicSearch` operation of the first service. Then, it stores the result into a variable and invokes `getItemsCount` operation of the same service to get the number of found items. If the returned value is equal to `"NotZero"`, the `addItems` operation of the `CartService` is invoked. The selected items (received using a `Receive activity`) by the customer are the input of this operation. The returned total price is stored. After receiving the email and the password of the customer the `signIn` operation is invoked. If the identification is passed successfully, an invocation to the `DeliveryService` is done. After that, the stored total price is used as input for invoking the `PaymentService`. At last, the `sendMail` operation is invoked with the necessary data.

6.6 Summary

In this chapter, we presented our approach of Web service orchestration and choreography generation. We have presented an algorithm that allows to generate choreographies starting from the method invocations located in the source code of the generate individual Web services. The presented algorithm, shows that the manual identification of these collaborations and the creation of Web service requests are too complicated tasks and time-consuming, especially, for a large source code and a great number of the generated Web services. The second algorithm that is presented in this chapter, allows to generate service orchestrations as BPEL processes starting from Web user interfaces navigations. The Web navigation documents are extracted from the source code (if they are not available as separate document such as JSF `faces-config` files). The presented algorithm deals with several aspects, namely, the creation of navigation paths, the creation of a WSDL description for each BPEL process, the identification of the involved Web services in the orchestrations, the creation of the necessary BPEL activities and partnerlinks elements, identification of the input and output variables, the identification of the variables that the client must provide via a receive activity, the matching of

variables' names (based a syntactic similarity) in the arguments of the operation to be invoked, and dealing with cycles. At the end, the BPEL processes are saved in files of XML-based format. Hence, we have shown that our approach alleviates in particular the developer from the complexity of identifying manually the Web navigations and preparing them to be considered as business logics for these orchestrations. In addition, we free the developer from the complexity of creating BPEL processes and their WSDL descriptions manually. (S)he does not need to deal with the complex XML data in BPEL and WSDL files.

The next chapter shows how to recover high level specifications starting from the source code of the generated Web service compositions.

CHAPTER 7

Recovering Architectures from Service Oriented Systems

> Models are abstractions that portray the essentials of a complex problem or structure by filtering out non-essential details, thus making the problem easier to understand ... Abstraction is a fundamental human capacity that permits us to deal with complexity ... We build models of complex systems because we cannot comprehend such systems in their entirety. There are limits to human capacity to understand complexity ... Models help us organize, visualize, understand and create complex things.
>
> Terry QUATRANI

7.1 Introduction

In this chapter, we present our recovery approach of service oriented architecture. In this approach, we aim to derive high level specifications from the source code of (Web) Service-Oriented Applications (namely the source code of the generated Web services using our approach in Chapter 5 and Chapter 6). In this way, we provide a useful support for maintenance developers and for the third party developers in order to understand the target service oriented system and make their evolutions easily. Our approach is based on two main steps: First, we generate BPMN specifications that represents the behavior of the system. We chose BPMN standard language because it provides a rich graphical notation for choreography modeling [Decker *et al.*, 2008]. Second, we generate architectures expressed with the SCA specifications [Beisiegel *et al.*, 2009]). These architectures represent the structure of the system using a set of components that are connected via explicit interfaces stating the provided and the required services.

We studied the application of our approach on two kinds of service oriented applications. First, we generate the service architecture from the source code of a Web service choreography. Second, we apply the approach on the OSGi (Open Services Gateway Initiative) [McAffer *et al.*, 2010] applications which are good examples of large Service Oriented Applications. This choice allow us to scale to real-world SOA applications (such as: Equinox Framework, Eclipse E4, and Eclipse Memory Analyzer Tool), where the number of the services is more than thousands.

7.2 Recovering Service Architectures from (Web) service Choreographies

The recovering of service-oriented architectures is a reverse engineering process which extracts high level specifications from the source code of (Web) service choreographies. A choreography of (Web) services is a group of services published by some providers that collaborate to form some consistent overall process. This collaboration is implemented by simply formulating requests from a service requester(client) to a service provider. The ultimate goal of our approach is to extract the hidden choreographies in order to help developers to understand the whole behavior of their applications, before starting to apply changes.

Overall, we first parse the source code of a Web service-based application to generate an abstract syntax tree. This latter is a tree representation of the abstract syntactic structure of the source code. It is created in conformance with Eclipse's ASTParser specification. After that, we transform this representation into behavioral and structural service oriented architectures using a set of transformation rules that we have defined. The recovering approach of BPMN models from Web service choreographies and the defined transformation rules have been validated in the publication [Kerdoudi *et al.*, 2016].

7.2.1 From (Web) services elements to BPMN elements

The generation of BPMN models from Web service Choreographies is based on the following set of rules which are defined as mappings between the source code elements and the BPMN elements.

1. For each choreography (collaboration between service providers and service clients), we create a BPMN model.

2. Each participant (service provider or service client) in the choreography is modeled with a `Pool` element[1]. This `Pool` is used as a container of the activities that are performed by the participant. The activities represent mainly the exchanges of the provided service (or service client) with other participants.

3. If the participant is a service client thus, it must contains one or several (Web) service invocation(s). Each service invocation is modeled as a `Task`[2] element to be added to the created `Pool` element of the service client. This `Task` element is a kind of activity within BPMN. It is an atomic Activity within a process flow.

4. The set of `Tasks` within a `Pool` are connected as a sequence (in the same order of their appearance in source code) using the `Sequence Flow` element.

5. If the participant is a service provider and its provided service has no requests for other services, the provided service is modeled as a `Service Task`[3] element to be added to the created `Pool` element of the service provider. The `Service Task` is a kind of Tasks within BPMN used to represent a kind of service, which could be a Web service.

6. If the participant is a service provider and its provided (Web) service implements requests to other services, the service is modeled as a `SubProcess`[4] element to be added in `Pool` element. The `SubProcess` element can be a white box or a contour which shows a lower-level process, which is executed by the service provider. Each request to an external service is identified and modeled as a `Task` element to be added as an inner activity of the `SubProcess` element. The set of `Tasks` within a `SubProcess` are connected as sequence using the `Sequence Flow` element.

7. The connection between a `Task` element (which is generated for a service request) and the `Service Task` (or the `SubProcess`) which located in separate pool is done via a

[1] A Pool is the graphical representation of a Participant in a Collaboration.
[2] A `Task` is a rounded corner rectangle which is drawn with a single thin line.
[3] A Service Task shares the same shape as the Task, which is a rectangle that has rounded corners, with a graphical marker in the upper left corner of the shape that indicates that the Task is a Service Task.
[4] A `SubProcess` is an Activity whose internal details have been modeled using Activities, Gateways, Events, and Sequence Flows.

`Message Flow` element. The `Message Flow` element is used to represent the message sending or receiving between the client participant and the service provider participant.

8. Each control statement (for example, a conditional statement) containing a service invocation is modeled with a diverging `Exclusive Gateway` (Decision) element[5]. It is used to create alternative paths within a process flow, only one of the paths can be taken. Each path is targeted to an activity element or to a default path (which can be the end of the process). The decision of the `Exclusive Gateway` can be a question or the conditional expression which is extracted from the source code (for example, the condition expression of a conditional statement). The path where the answer to this question is true is targeted to the `Task` element created for the service invocation. The two elements (`Exclusive Gateway` and `Task` elements) are connected using a `Sequence Flow` element.

9. If the service invocation statements are located within a loop statement, the loop is represented with a `Loop Task` element. It has the same form as the `Task` with a loop marker in the medium. And, we move the `Task` elements created for the service invocation statements into the `Loop Task` element.

7.2.2 Example of generating BPMN models from a Web service choreography

Let us return to our example presented in section 6.3. We show here how to create a BPMN model (behavioral model) from the generated Web service choreography (which can be seen at code level only). Figure 7.1 shows the generated BPMN model[6], that represents the participant services which are involved in the payment activity in this example.

As we can see in this model, all the hidden choreographies are extracted and we can easily understand the behavior of the overall composition. This is due to our technique of reverse engineering, since only operation invocations and the control or loop statements that contain operation invocations are extracted. In this way, the developer does not need to know all the details which could be found in a large source code. Only participants (Web services) in the choreography and their exchanged messages are modeled.

By extracting these models, we believe that thanks to the high level of abstraction provided by the choreographies, evolving these applications will be simpler. Indeed, we help the developer in choosing the well suited position in the code in order to apply the necessary changes to implement an evolution scenario.

[5] It has the form of a diamond with a marker inside that is shaped like an "X."

[6] The generated model is updated and validated manually by grouping the `Pools` which are of the same category and giving more readable names to `Pools`, `Lanes` (sub-partition within a Pool) and `Activities`.

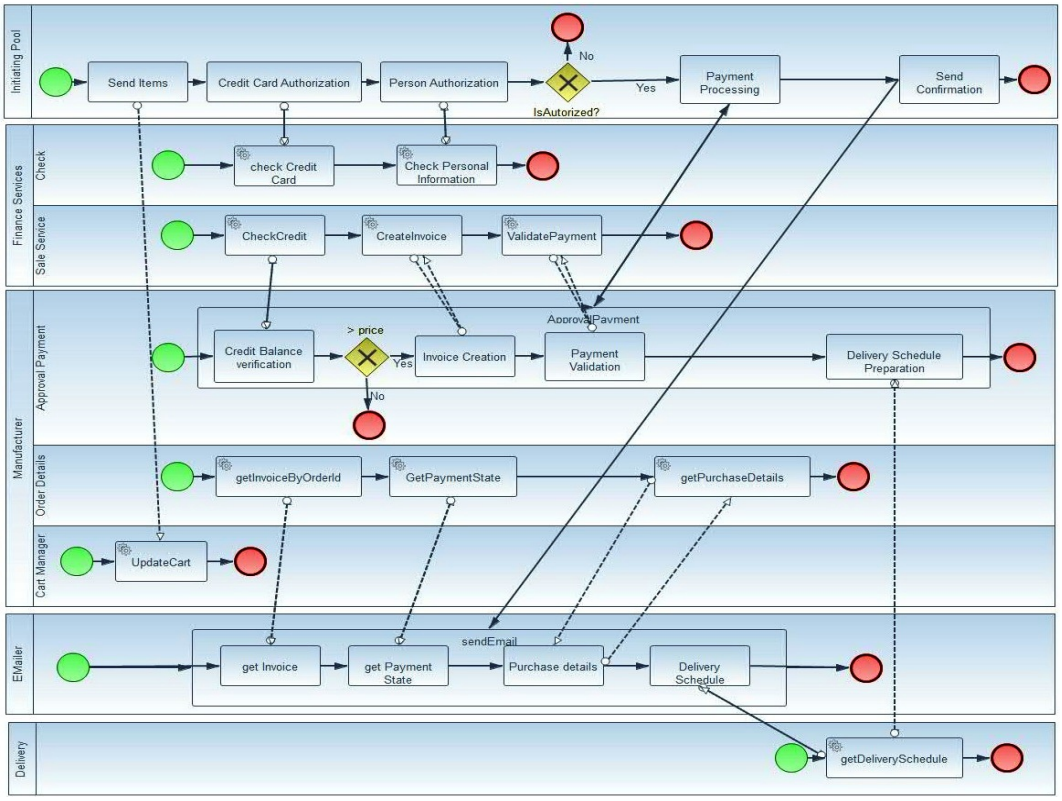

Figure 7.1 : A BPMN model represents a service choreography to accomplish the payment activity.

Let us suppose the following evolution scenario for the payment activity choreography. We need to add a new Web service that enhances the security of the payment process. This service can be a program that sends to the client a validation code with an SMS on her/his smartphone. The client introduces the received code and the application checks its validity. On looking at the generated BPMN model, the developer can clearly decide to add an invocation to the new service after the `Credit Card Authorization` task in the initiating Pool.

7.2.3 From (Web) services elements to SCA elements

The creation of SCA models is based on a set of transformation rules that we have defined as mappings between the source code elements to SCA elements.

1. For each choreography (collaboration between service providers and service clients), we create a Service Component Architecture instance, which is represented by an SCA composite.

2. Each participant (service provider or service client) in the choreography is modeled with an `SCA Component` element. This `SCA Component` is used as basic element of business function in an SCA assembly, which is combined into complete business solutions by an `SCA composite`. Components provide and consume services. They are declared as sub-elements of an `SCA Composite`.

3. If the participant is a service client thus, it must contain one or several (Web) service invocation(s). All the identified operation invocations of the same service are modeled as an `SCA Component Reference`[7] element to be added to the created `SCA Component` element for the service client (see Rule 2). This `SCA Component Reference` element represent a required interface (service) for the service requester participant. The name of the `SCA Component Reference` must be identical to the name of the interface of the invoked service.

4. If the participant is a service provider then, it is modeled as an `SCA Component` (see Rule 2). This `SCA component` must has an `SCA Component Service` element[8], which represents the interface of the offered (Web) service.

5. If the participant is a service provider and the implementation of its provided (Web) service contains a request(s) to other service(s), the service provider is modeled as an `SCA Component` (see Rule 2) and its offered service is modeled using an `SCA Component Service` element (see Rule 4). And, all its operation invocations of the same service are modeled as an `SCA Component Reference` (see Rule 3).

6. Each invocation from a service client to a (Web) service (or from a (Web) service to another Web service) is modeled structurally as an `SCA Wire` element that connects the `SCA Component Reference` element (from `SCA Component` of the service requester) and the `SCA Component Service` element (from the `SCA Component` of the service provider).

7.2.4 Example of Recovering SCA models from a Web service choreography

Let us consider the same example which is presented in section 6.3. We show here how to create an SCA architecture (structural view) from the generated Web service choreography. Figure 7.2 depicts the generated SCA architecture for the payment application[9]. First, for each Web service involved in the payment activity, we create an `SCA component`. The name of all the generated `SCA components` from this example are as follows: `PaymentProcess`, `Check`, `Sale`, `ApprovalPayment`, `OrderDetails`, `CartManager`, `Emailer`, `Delivery`. Each of these components has a provided interface that represents the interface of offered Web service. This interface is modeled with an `SCA Service` element. The invocation of an operation located in this

[7] An `SCA Component Reference` has a chevron shape with a purple color
[8] An `SCA Component Service` has a chevron shape with a green color
[9] This model is opened and visualized graphically with the eclipse SCA Composite Designer Framework.

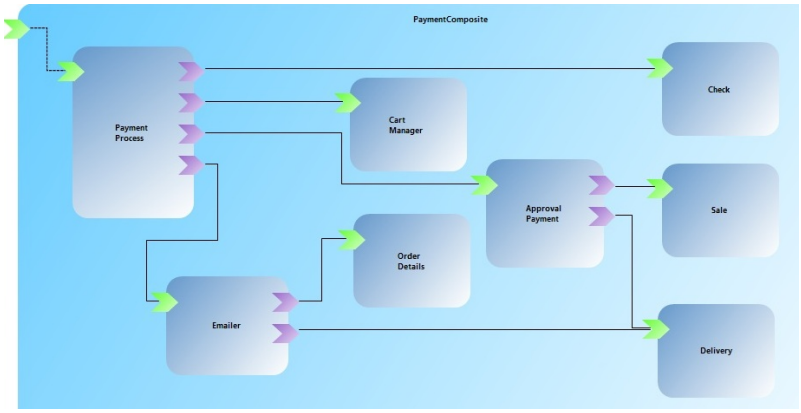

Figure 7.2 : Service Component Architecture of the Payment Application

Web service is represented by an `SCA Reference` element in the `SCA Component` that represents the service requester (caller). This `SCA Reference` has the same name as the Web service interface's name.

To implement the previous evolution scenario, this architecture indicated to us that, we need to update the source code of the `PaymentProcess` component by adding the concerned Web service request. The invoked new Web service (SMS sender) could be provided by a separate service provider. In this case, after such evolution, we can see in the new generated architecture, it contains a new SCA component connected to the `PaymentProcess` component. In the case of the SMS sender service is implemented by one of the existing components such as the `CartManager component`, after such evolution, we can see an `SCA component Service` element is identified and added to this component. And, it is connected to the `PaymentProcess` component.

7.3 SCA Component grouping and SCA Composite generation model

For a large (Web) service oriented applications where the number of services is large (hundreds and thousands), the generated SCA architecture should have a great number of connections between components. This makes the understanding of such kind of system difficult. To deal with this issue, in our approach, we help the maintenance engineer by providing additional models at high level of abstraction.

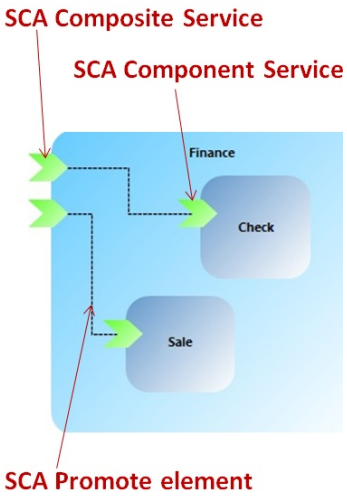

Figure 7.3 : Architecture of Finance SCA Composite

7.3.1 Grouping SCA Components into an SCA composite

We proposed to generate several models of different levels of abstraction. Indeed, in addition to the generated flat organization[10] of the SCA components, we create new SCA models (a set of SCA composites) where, we group the SCA Components of the same category into an SCA Composite. Each SCA Composite is a separate model. For example, Figure 7.3 shows that, the `Check` and the `Sale` SCA components (of the previous example) are grouped together to form one SCA Composite has the name `Finance`. Figure 7.4 shows that, the `CartManager`, the `OrderDetails` and `ApprovalPayment` SCA components are grouped together to form one SCA Composite has the name `Manufacture`.

Now, if an `SCA Component` of `SCA Composite 'A'` provides a service that is required by another `SCA component` located an `SCA Composite 'B'`(outside of `SCA Composite 'A'`), we create an `SCA Composite Sevice` element in the `SCA Composite` 'A'. Inside the `SCA Composite 'A'`, we create an `SCA Promote` element[11] that connects the created `SCA Composite Service` and `SCA Component Service` of the component that provides this service. In this way, we make this provided service accessible from the outside of this `SCA Composite`. In the other side, we create an `SCA Composite Reference` element in the `SCA Composite` 'B'. Inside the `SCA Composite 'B'`, we create an `SCA Promote` element that connects the created `SCA Composite Reference` and `SCA Component Reference` of the component that requires this service.

For example, in Figure 7.4, we created an `SCA Composite Sevice` element for the provided

[10] This means that all the components and their structural connections are modeled in one model.

[11] SCA promote element means that the composite service is actually provided by one of the components within the composite

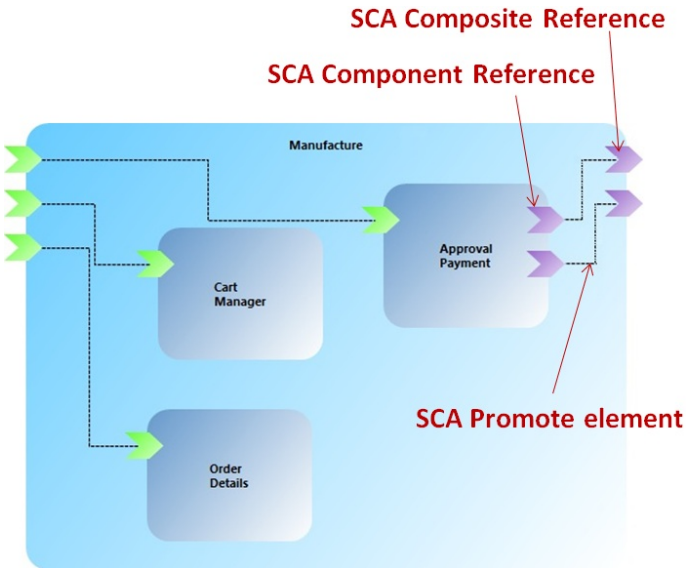

Figure 7.4 : Architecture of Manufacture SCA Composite

service by the `Sale SCA component`, because, it is required by `ApprovalPayment Component` (which is located in the Manufacture Composite). A promote element (shown in dashed lines) is created to connect this `SCA Composite Sevice` and the `SCA Component Sevice` inside the `Finance` Composite. Figure 7.4 shows an `SCA Composite Reference` element is created in the `Manufacture` composite, which constitutes the `ApprovalPayment Component`. And, a promote element inside the `Manufacture` composite is created to connect `SCA Composite Reference` and `SCA Component Reference`.

In the current implementation, the grouping of components is performed semi-automatically by assisting the developer in order to give the best grouping of SCA components based on the category of the provided services and developer knowledges. In the near future, we plan to use a hierarchical clustering technique such as Weighted Combined Algorithm [Maqbool et Babri, 2004], where, we represent each component by a cluster, and we compute the pair-wise similarity between all the clusters and then, we combine the two most similar clusters into a new cluster. This is repeated until all the elements will be clustered or the desired number of clusters is achieved.

7.3.2 Creation an SCA composite starting from a set of SCA composites

Once all the SCA Composites are created, we generate a new model (of high level of abstraction) that groups together these SCA composites to form an SCA Architecture for the whole application. Each `SCA Composite` (which is created previously by grouping components) is modeled as an `SCA component` in this new architecture. The dependencies between SCA composites are

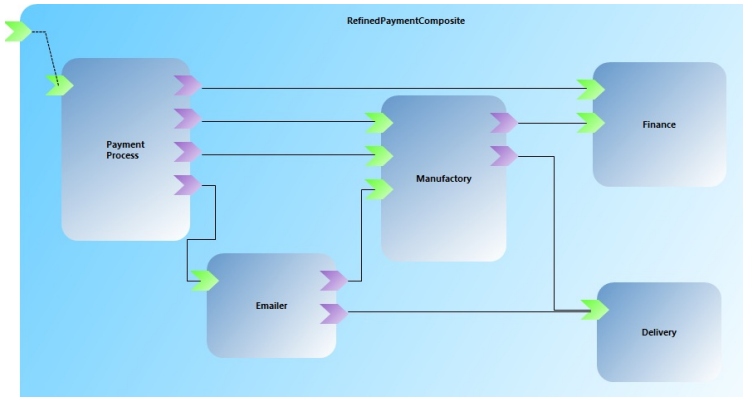

Figure 7.5 : Refined Service Component Architecture of the Payment Application

identified based on the name of the provided and the required services. These dependencies are then modeled in this architecture using the `SCA Wire` elements. In this way, we provide an architecture of high level of abstraction that represents the application by hiding the details of each SCA composite (which is modeled in separate model with its components). Figure 7.5 depicts the generated global architecture of the payment application. In this architecture, the `Manufacture` and the `Finance` composites are modeled as SCA components.

7.4 Recovering Service architectures from OSGi-based Applications

We have presented in the previous sections how to generate BPMN and SCA specifications from the source code of a Web service choreography. We have introduced a set of transformation rules that allow the mapping from the source code elements into BPMN elements and into SCA elements. These rules are extensible and easily testable and quickly debuggable. They are defined to be applied to any service oriented system which has a set of participants that collaborate together to form some consistent overall process that represents a service choreography. As we have aforementioned, this collaboration is implemented by formulating requests from a service requester (client) to a service provider. This service provider can also formulate one or several requests to other service providers. Our reverse engineering process aims to extract these collaborations by identifying the statements that represent the service invocations from the service clients to the service providers.

In this section, we illustrate how to use the previous transformation rules to recover the BPMN and SCA models starting from another kind of service oriented systems, which are applications created and executed under the OSGi Framework. This allows us to study the application of our recovery process on large scale applications. Before presenting the details of the service-oriented architecture generation from the OSGi applications, we introduce with a concrete example a brief description of the OSGi component model.

7.4.1 OSGi Component

The core specification of OSGi defines a component model and a Framework for executing components. A component in OSGi is known as a bundle [Hall *et al.*, 2011]. Each bundle is defined by a single JAR file which packages the modules and a manifest file which contains the extra meta-data. Listing 7.1 shows the meta-data in a the Manifest of a bundle. The manifest file declares which of the packages are externally visible using "export-package". The manifest file can declare explicitly which are the bundles it depends on using "import-package" or "require-bundle". They are functionality consumed by the bundle and they are provided by others bundles. The "require-bundle" is used when the bundle requires another bundle. The first bundle has access to all the exported packages of the second. We consider in our work the imported packages as the required interfaces of the component and the exported packages as the provided interfaces.

```
1 Bundle-ManifestVersion: OSGi specification (use the value 2 for OSGi release 4)
2 Bundle-SymbolicName: The only mandatory forconsidering a JAR as a OSGi bundle. This
      is the unique name of the bundle
3 Bundle-Name: A name easy to read by humans (without spaces)
4 Bundle-Version
5 Bundle-Activator: the 'Activator' class
6 Import-Package: List of required packages by the bundle
7 and which are provided by other bundles
8 Export-Package: List of packages to be exported
9 Require-Bundle: List of bundle names
```

LISTING 7.1 : Metadata (Headers) in the Manifest

The class « Activator » is implemented by the bundle. It extends the `org.osgi.framework.BundleActivator` class and contains methods ("start" and "stop") related to the life cycle of a bundle. The "start" method is called when the bundle is started so the Framework can perform the bundle-specific activities necessary to start this bundle. The "stop" method is called when the bundle is stopped. The two methods are invoked automatically by the Framework.

7.4.2 The OSGi Framework

The Framework forms the core of the OSGi specifications. The functionality of the Framework is divided in the following layers: Security Layer, Module Layer, Life Cycle Layer, Service Layer, Actual Services (this layering is depicted in Figure 7.6) [Alliance, 2014]. The Security Layer is based on Java 2 security. The Module Layer defines a modularization model for Java. The Life Cycle Layer provides a life cycle API to bundles. This API provides a runtime model for bundles. The **Service Layer** provides a dynamic, concise and consistent programming model for Java bundle developers. It introduces a `Service Oriented Programming` model. It is like **SOA in**

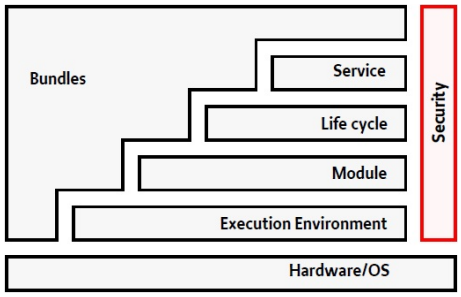

Figure 7.6 : OSGi Framework Layers [Alliance, 2014]

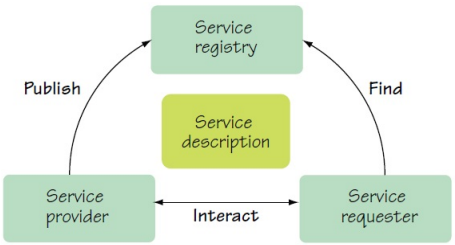

Figure 7.7 : The service-oriented interaction pattern [Hall *et al.*, 2011]

Virtual Machine, which implements the `Service Locator Pattern`. The main concepts revolve around the service-oriented publish, find, and bind interaction pattern. A bundle (service provider) can publish their exported interface as services into a **Service Registry** (the "start" method that is implemented by the class « Activator » (called also: `ServicePublisher`) can be used to register services in the **Service Registry** or to allocate any resources that the bundle needs), while a service client (another bundle) search the registry to find available services to use (see Figure 7.7). This bundle receives from the Service Registry a reference of an instance service implementation.

7.4.3 Example of an OSGi-based application

We take as example a simple E-Mailer OSGi-based application that have been implemented as set of bundles (components) that collaborate together to prepare and send an email. Here we give the details of each component in this application and their provided and required services. The components that compose the application are described as follows:

- `Emailer Component`: it is the main component in this application. It collaborates with the other components in order to provide the email preparing and sending services.
- `AddressBook Component`: this component provides services such as: storing, accessing, sorting and updating entries called contacts. Each contact entry usually consists of a few

standard fields (for example: first name, last name, company name, address, phone number, e-mail address, fax number). The services provided by this component are invoked from the `Emailer Component`.

- `TextEditor Component`: it provides several services for editing the message to send, such as: get the edited message, receive user input text, set the font, add a numbered list, and insert a picture. These services are invoked from the `Emailer Component`.
- `SpellChecker Component`: it provides services to check the edited input text. The services could be: fix the spelling errors or provide a set of suggestions to fix the errors, registering/unregistering a new dictionary. These services are invoked from the `TextEditor Component`.

The Listing 7.2 depicts the interface that is exported as a service (contains a set of methods) by the `SpellChecker` component.

```
1 package dz.biskra.univ.spell.checker;
2 public interface SpellCheckerService {
3   public void registerDictionary(Dictionary dictionary);
4   public void unregisterDictionary(Dictionary dictionary);
5   public boolean spellCheck(String word);
6   public Vector<String> getSuggestions(String word);
7 }
```

LISTING 7.2 : The Interface of the service provided by the SpellChecker component

The Listing 7.3 depicts an excerpt of the code of the class « Activator » that publishes the service `SpellCheckerService` into the Service Registry, thanks to the "start()" method. The "start()" method uses the provided bundle context and their implemented method "**registerService(...)**" to register the interface as a service in the **Service Registry**.

```
1 public class Activator implements BundleActivator {
2  ServiceRegistration registration ;
3  public void start(BundleContext ctx) throws Exception {
4    registration = ctx.registerService(SpellCheckerService.class.getName(), new
         SpellCheckerServiceImpl(), null);
5  }
6  public void stop(BundleContext bundleContext) throws Exception {
7    registration.unregister();
8  }
9 }
```

LISTING 7.3 : The Activivator of the SpellChecker component

In the first parameter, we need to provide the interface name that the service implements, followed by the actual service instance (« new SpellCheckerServiceImpl()»), and finally the service properties (see Line 4 in Listing 7.3).

We have seen how to register a service. Now, we present how to discover this service (provided by `SpellChecker component`) from another bundle client (for example, the `TextEditor component`). In order to use a service object and call its methods, a bundle must first obtains a **ServiceReference** object. The `BundleContext` interface defines a number of methods a bundle can call to obtain `ServiceReference` objects from the Framework. The Listing 7.4 shows a simplistic client code that could be implemented by the `TextEditor component`. This code represents an excerpt of the `getMessage` method implementation [12]. This code uses the provided bundle context and their implemented methods "**getServiceReference(...)** and **getService(...)**". The `getServiceReference(...)` method is used to obtain the reference of the registered service (`SpellCheckerService`) (see Line 8). It has one input which is the class name with which the service was registered. The `getService(...)` method returns the service object from the `Service Registry`(see Line 9). It has one input represents the service reference.

```
1  package dz.biskra.univ.text.editor;
2  import org.osgi.framework.*;
3  import dz.biskra.univ.spell.checker.SpellCheckerService;
4  public class TextEditorServiceImpl implements TextEditorService {
5   public String getMessage() {
6     BundleContext ctx = getContext();
7     String msg= getUserInput();
8     ServiceReference ref = ctx.getServiceReference(SpellCheckerService.class.getName
          ());
9     SpellCheckerService spellChecker = (SpellCheckerService) ctx.getService(ref);
10    //Tester ref!=null
11    Boolean isCorrect = spellChecker.spellCheck(msg);
12    if(isCorrect) { return msg;}
13    else {....} // fix errors
14   }
15 //...
16 }
```

LISTING 7.4 : An excerpt of a simplistic client code of the SpellCheckerService

7.4.4 OSGi application Parsing

In order to apply the previous transformation rules on the OSGi-based applications, we need to identify from their source code and the manifest files the following elements: (i) the partic-

[12] This method is provided later as a service by the `TextEditor component` via the interface `TextEditorService`.

ipants (bundles) that provide services (service providers) (ii) the participants that use services (service clients), (iii) the name of the provided and the required services, and (iv) the invocation order of these services (this corresponds to the order of their appearance in the source code).

Provided Services and Service Provider identification

We distinguish two types of provided services by a bundle as follows:

- `Provided Services via the Service Registry`: this kind of services is identified by the parsing of the source code of each bundle. The parsing focuses on finding the statements that allow the bundle to *register services* in the Service Registry. For example, in the E-mailer application (in Section7.4.3), we have the `SpellChecker` bundle which has registered a service in the Service Registry via the statement: "ctx.registerService(...)", (see Line 4 in Listing 7.3). So, the `SpellChecker` bundle is considered as a `Service Provider`. The name of the provided service is identified from the input parameters of the *registerService(...)* method (the *SpellCheckerService* in this example).
- `Provided Services via the Manifest file`: this kind of services is identified by the parsing of the manifest file of each bundle. As explained in the Section7.4.1, we consider each exported package as a provided service while the name of this service corresponds to the name of this package.

Required Services and Service Client identification

We distinguish two types of required services as follows:

- `Required Services via the Service Registry`: this kind of services is identified by the parsing of statements that are used to *discover* the services from the Service Registry. For example, in the E-mailer application, we have the `TextEditor` bundle which uses the service `SpellCheckerService` that is provided by `SpellChecker` bundle via the statements "*getServiceReference(...)*" and "*getService(...)*". So, we consider the `SpellCheckerService` as a required service for the `TextEditor` bundle, while the latter is considered as a Service client.
- `Required Services via the Manifest file`: the imported packages are considered in our approach as required services, while the name of each service corresponds to the name of imported package.

Besides, in a bundle, we can have some services that are imported or their service references are obtained (via the *getServiceReference(...)* statements), but these services are not really invoked (or are not instantiated) in the source code of the client service. This kind of services is called

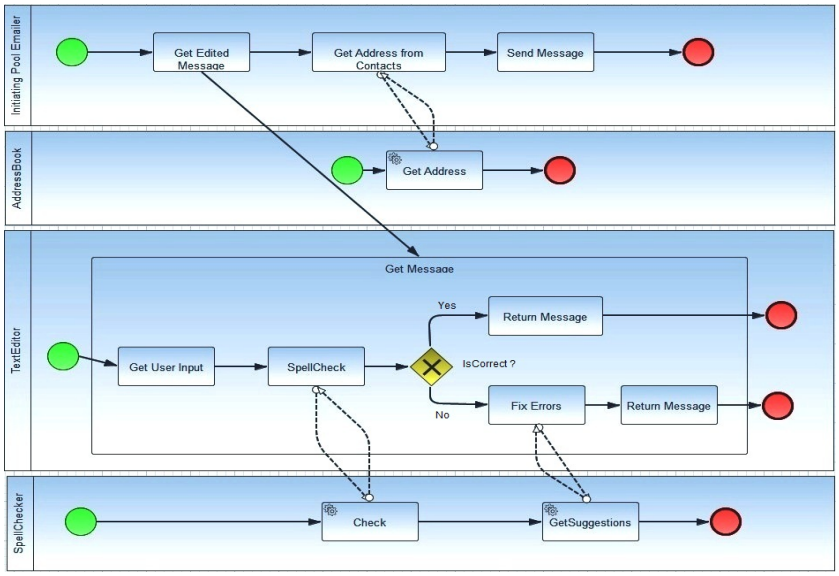

Figure 7.8 : Recovered BPMN Architecture from the Emailer application

in our approach: **Potential Required Services**. We call the connector that links this bundle with the bundle that provides the Potential Required Service: a **Future Potential Connector**. In the current implementation, we distinguish between the two kinds of the required services (or connectors) by using a documentation that is added (as an additional tag in the XML file) in the `SCA Component Reference` element and the `SCA Wire` element of the generated SCA architecture. In the near future, we plan to extend the Apache Tuscany SCA implementation of the SCA specification for doing this distinction by using different colors for these elements. Moreover, in this extension, we will distinguish between the services that are provided (or required) via the Service Registry and those that are provided (or required) via the Manifest file.

This distinction between "potential" elements in an architecture and effective ones enables to recover architectures that can be visualized with some level of detail (LoD) as in 3D or game graphics. An architect can visualize a detailed architecture, with all its elements: potential and effective ones. This will enable her/him to make a deep analysis of this architecture. She/He can also visualize a general recovered architecture to have an overview of the structure of the analyzed system. This latter case is particularly interesting if the system is very complex

7.4.5 Recovering the BPMN Architecture from the E-Mailer application

Figure 7.8 depicts the recovered BPMN model for the E-Mailer OSGi-based application. This model is recovered using the same transformation rules which are presented in Section 7.2.1. The participants here represent the bundles (bundles provide services and bundles consume services). We have four bundles collaborate together as an overall process in order to achieve a

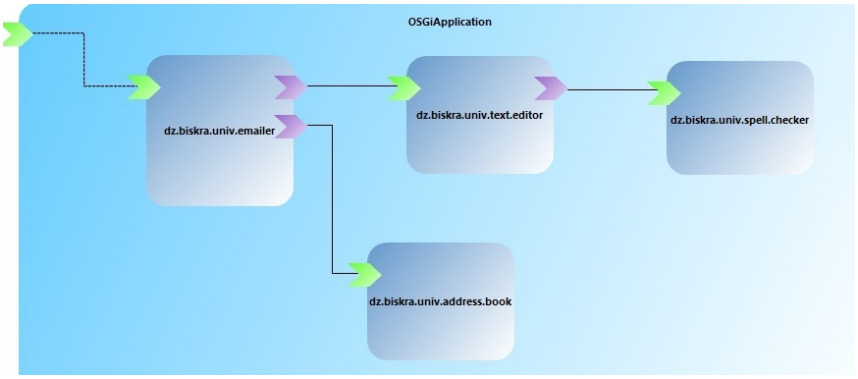

Figure 7.9 : Generated SCA model that represents the E-Mailer OSGi application

common goal (sending an email). A BPMN instance is created by applying the Rule 1 to our example. Four `Pool` elements are created (AddressBook, TextEditor,... in Fig. 7.8) by applying the Rule 2. The bundle `TextEditor` contains an invocation to the service `SpellCheckerService` which is provided by the `SpellChecker` bundle (see Lines 8 to 11 in Listing 7.4). The invocation is modeled as `Task` element added to the `TextEditor Pool` (Rule 3). The invoked service is represented with a `Service Task` in the `SpellChecker Pool` (Rule 5). This service is identified by the parsing of the source code of the `SpellChecker` bundle, and extracting the call to the `registerService(...)` method (see Line 4 in Listing 7.3). We do the same work for modeling all the interactions between bundles. The `getMessage()` service is used by the `Emailer` bundle, and the implementation of this service contains requests to the services (check and getSuggestions) provided by the `SpellChecker` bundle. In this case, the `getMessage service` is modeled as a `SubProcess` element (Rule 6). The `Exclusive Gateway` element is created in the `TextEditor Pool` by applying the Rule 8.

7.4.6 Recovering the Service Component Architecture from the E-Mailer application

Figure 7.9 depicts the recovered SCA model for the E-Mailer OSGi-based application. We used for creating this architecture the same transformation rules that are presented in Section 7.2.3. An SCA instance is created from the collaboration in this example. Four SCA components are created for each bundle (Rule 2) (see Fig. 7.9). All the four bundles register services in `Service Registry` by calling the `registerService(...)` method. So, an `SCA Component Service` element is added to all the created SCA Components (Rule 4). The invocation of the `getMessage` service (provided by the `TextEditor` bundle) from the `Emailer` bundle is modeled by `SCA Component Reference` element in the `Emailer SCA Component`, and an `SCA Wire` element is created to connect the created `SCA Component Reference` and the `SCA Component Service` elements. We do the same work for the service invocations which are identified from the other bundles. Finally, we obtain the architecture illustrated in Figure 7.9.

7.5 Summary

In this chapter, we have presented a reverse engineering approach for deriving a service oriented architecture from the source code of the Web service choreographies. We generate the structural and behavioral views based on a set of transformation rules that we have defined. In our recovery approach, we focused on identifying the service invocations. In this way, we hide from the developer the unnecessary details which could be found in a large source code. Only participants in the choreography and their exchanged messages are modeled. Hence, the developer can easily understand the interactions in the overall composition of Web services.

We have shown that the defined transformation rules are extensible and can be applied to any service oriented systems (not necessarily Web services). We have tested these rules on another kind of service-oriented systems which are implemented within the OSGi Framework.

In next chapters, we present the details of the implementation of the proposed approaches in this thesis. After that, we present the results of the conducted experimentation in order to evaluate and validate the proposed ideas in this thesis.

CHAPTER 8

Tools

Make it work.
Make it work right.
Make it work right and fast.

Edsger DIJKSTRA, Donald KNUTH, C.A.R. HOARE

8.1 Introduction

Both ideas of generating primitive and composite Web services from a Component-based Web application (Chapter 5 and Chapter 6) and Recovering Architectures from Service Oriented Systems (Chapter 7) are put into practice via two tools: **WSGen** and **ArchGen** respectively. The first tool allows generating and compiling Java Web services and BPEL processes. It generates also: the WSDL documents for each generated individual and composite Web service, the configuration files, and the axis archives for deploying these services. The second tool allows to generate high level specifications described in BPMN and SCA. The generated specifications are XML files that can be opened and visualized graphically in a dedicated Graphical Modeling Framework such as, Eclipse BPMN2 Modeler [Eclipse, a] and SCA Composite Designer [Eclipse, b]. This chapter presents also the functional architecture of WSGen and ArchGen, where, we describe the role of each component in these architectures. In addition, we show here what are the generated Web services and their compositions from an example of a Web application.

8.2 WSGen: A tool for creating primitive and composite Web services starting from Web components

We implemented the proposed solution as a tool called **WSGen**: **W**eb **S**ervice **Gen**erator. This tool receives, as input, archives of Web components and generates a set of primitive and composite Web services corresponding to these components. These Web services are deployed on a Web service provider according to the choices of the developer.

8.2.1 WSGen's Functional Architecture

As **WSGen** implements the multi-step process (presented in the Chapter 5 and Chapter 6), we have chosen the pipeline architectural style [Shaw et Garlan, 1996] for its implementation. Figure 8.1 depicts its abstract architecture. 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 by the **ArchiveParser** component to identify and filter (according to the OCL constraints defined by developers) the operations to be published in Web services. The **OpDistributer** component allows the distribution of operations on multiple services according to the two criteria discussed previously. The new organization of the operations is provided as output of this component. The **WSCreator** generates primitive Web services (Java classes) and the BPEL processes starting from the set of operations produced by the **OpDistributer** component. For each generated Web service (primitive and composite one), a WSDL document is generated. **WSDeployer** uses a Tomcat server associated with the Apache implementation support for Web services, Axis. This component generates a set of deployment

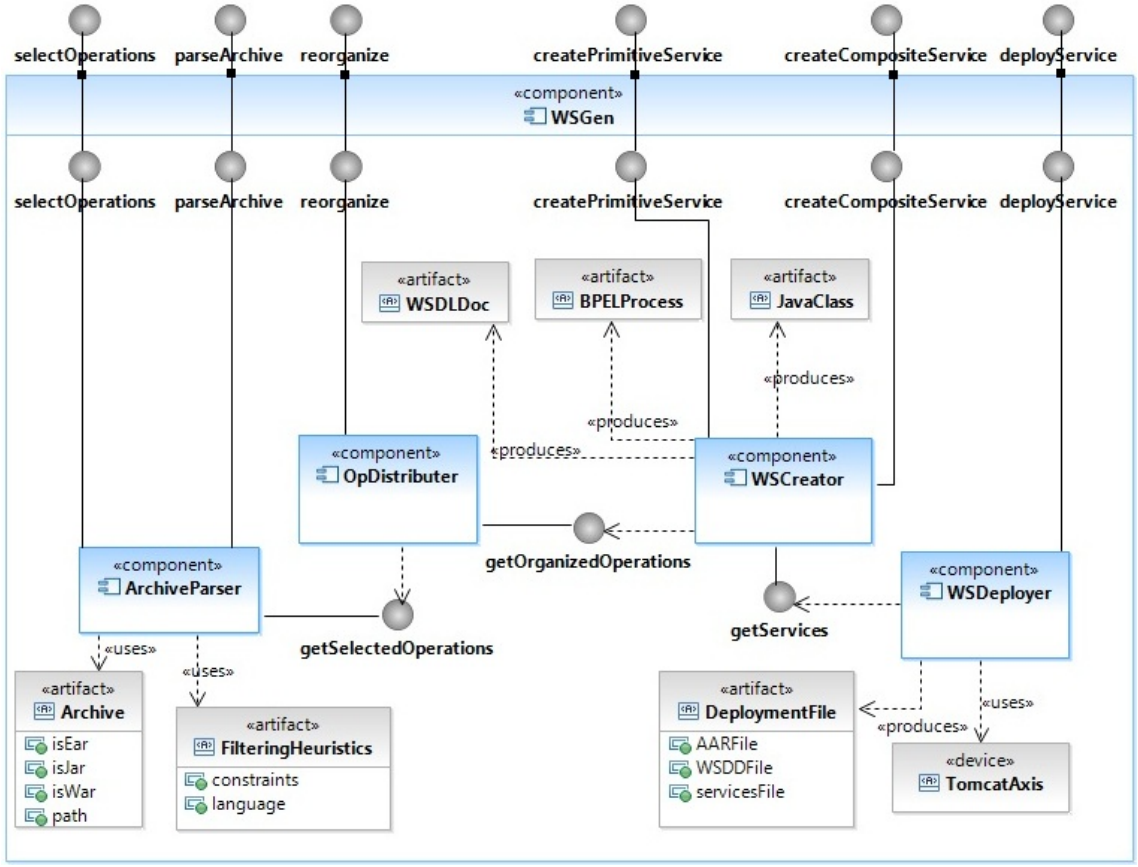

Figure 8.1 : The Abstract Architecture of WSGen

files (for example, Axis ARchive (AAR) files and Web Services Deployment Descriptor (WSDD)), which are used to deploy the desired Web services.

8.2.2 WSGen By Example

This section illustrates the application of the approach on an example of a Web application. Indeed, we explain how to create a Web service-oriented solution starting from a simulated version of a real-world Web application. The latter 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. This example has been used and validated in the publication [Tibermacine et Kerdoudi, 2012].

- 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.

8.2.3 Generated Primitive Web services

The transformation of this Web component generates the following set of primitive Web services [1]:

- `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.

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

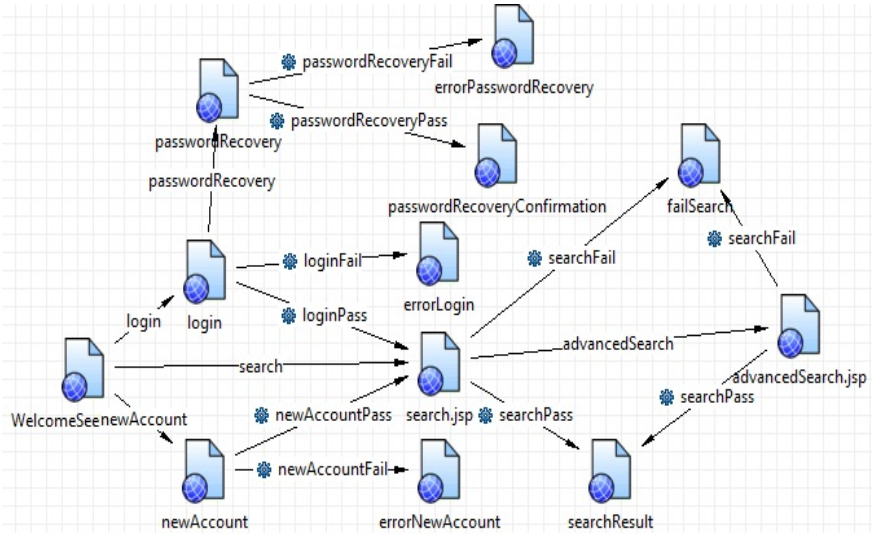

Figure 8.2 : Navigation Rules in the simulated Seekda Web Application

- 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.).

8.2.4 Generated Composite Web service

Figure 8.2 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 8.2 there are sixteen[2] navigation paths for the seekda web application. For each navigation path, a BPEL process is generated using the previous algorithm. Figure 8.3 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 a set of port types, through which it provides operations to clients. As depicted in Figure 8.3, 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 ex-

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

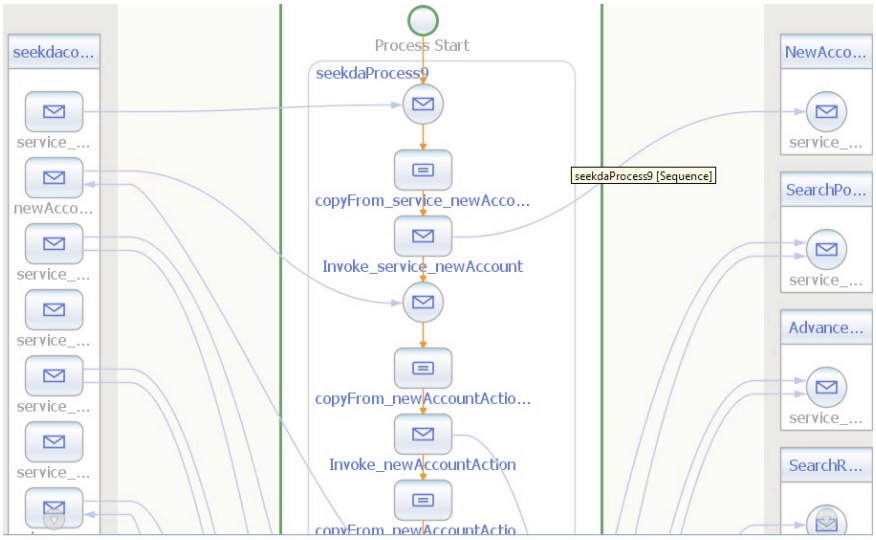

Figure 8.3 : Excerpt of the generated BPEL Process

ternal 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 8.2 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. They can build extensions of these services to provide more sophisticated solutions. For example, the tools implemented by our team [Azmeh *et al.*, 2011] 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.

8.3 ArchGen: A tool for recovering Service Architectures from the source code of Service Oriented Systems

In order to validate the idea of Recovering Architectures from Service Oriented Systems (Chapter 7), we have implemented a tool called **ArchGen**: **Arch**itecture **Gen**erator. This tool receives as input the source code of a service oriented application (namely the generated Web service application from Web components), and generates a set of BPMN and SCA models as high level specifications.

8.3.1 ArchGen's Functional Architecture

Figure 8.4 depicts the abstract architecture of **ArchGen**. The input of ArchGen are the source code of Java Web services or OSGi bundles' source code. Each Java class in this components is extracted and parsed. The parsing is performed using the **SourceCodeParser** component. An Abstract Syntax Tree (**AST**) representation of the source code is created by this component and provided as output. The **CollaborationIdentificator** component analyzes the AST in order to extract the set of collaborations and the involved participants (service clients and service providers). **BPMNGenerator** and the **SCAGenerator** components use these collaborations as inputs to generate respectively BPMN architectures (as "*.bpmn2" files) and SCA specifications (as "*.composite" and "*.composite_diagram" files). The **SCAGenerator** component provides (via the interface "createSCAComposites") an assistance to the designer to group SCA components into an SCA Composite. This grouping is based on the component's names and the designer knowledge. After that, the **SCAGenerator** component provides a functionality via the interface "createHLSpecification") allows to create a high level specification (as an SCA composite) by grouping these newly created SCA composites.

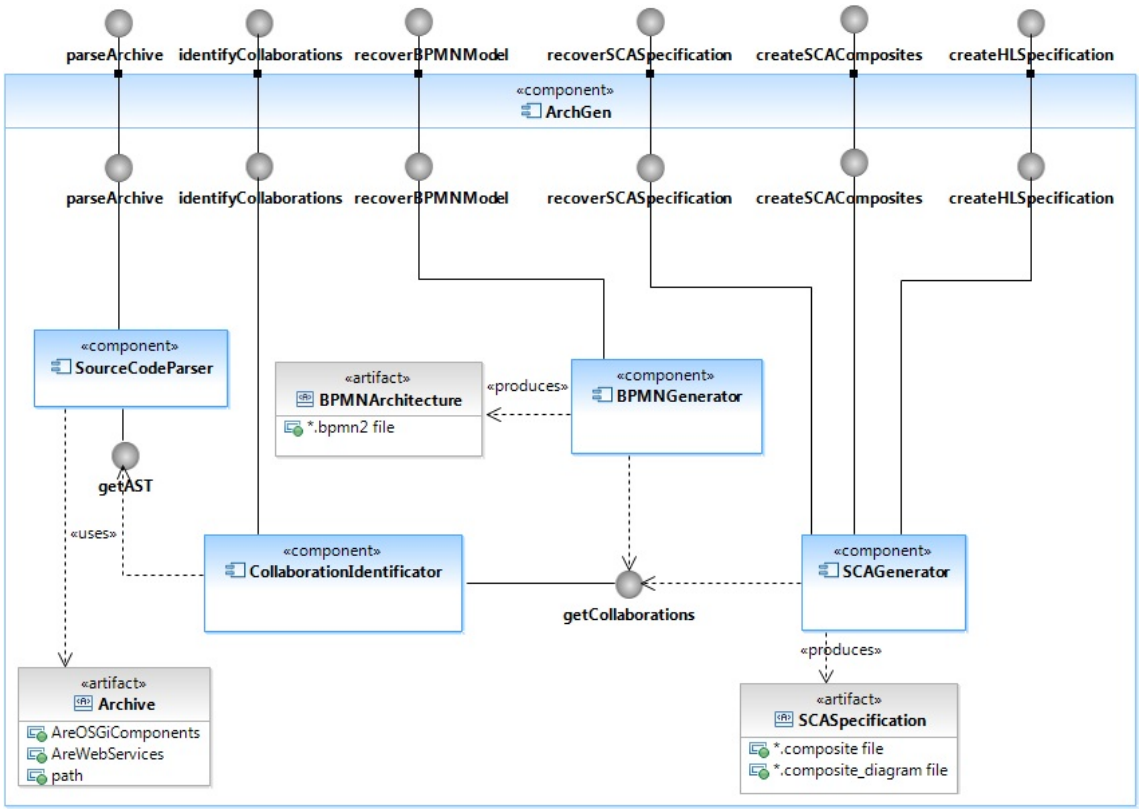

Figure 8.4 : The Abstract Architecture of ArchGen

8.4 Underling Technologies

WSGen and ArchGen have been implemented on the basis of the following technologies (among others):

Eclipse ASTParser tool : All tasks related to the parsing of Java source code and the identification of dependency between operations are implemented using the ASTParser tool. This tool is available in the Eclipse Java development tools JDT [Eclipse, 2001].

JaxMe Java Source framework : This framework [Apache, 2003] is used for generating Java source code. In WSGen, the API of this tool has been exploited to create new operations and Java classes implementing the exported services.

MDT OCL: This tool [Eclipse, 2009b] aims at providing a support to developers working with models containing expressions written in OCL. Such support includes the edition, code generation, interpretation, and debugging of the OCL constraints given for some underlying (Ecore or UML2) model. In our case, the OCL constraints are checked on an Ecore instance built starting from the operations' code.

HTMLParser: This tool [Derrick *et al.*, 2002] is a Java library used to parse HTML pages. WGen adapts this tool to parse JSP files and extracts some code portions to generate Web service operations. This allows to identify scriptlets, expressions, declarations and other JSP tags.

JspC: We use JspC to convert JSPs into Servlets. The generated classes are parsed to help us in creating input/output parameters. JspC is available in Tomcat Apache Project [Apache, 2005]).

Axis' Java2WSDL: This tool is provided by Apache with the Axis Web service support [Apache, 2004a]. WSGen uses Java2WSDL to generate WSDL files starting from the Java files containing the operations that implement the created Web services.

Xerces-J: This tool [Apache, 1999] provided by Apache as an XML parsing support is used to analyze the navigation documents in the Web components.

WSDL4J: The WSDL documents are parsed using this tool. WSGen uses WSDL4J API to create WSDL documents, which represent the interfaces of the generated primitive and composite Web services.

BPELWriter: This API [Eclipse, 2005] is a Java library used to generate BPEL processes. The process is instantiated by BPELFactory. This instance is registered by a resource using BPELResourceSetImpl, which in turn is used by an instance of the BPELWriter class. BPELWriter has methods for translating each of the BPEL elements and attributes from EMF objects into DOM objects. DOM objects are serialized to BPEL files.

XML Schema Definition Tool: This tool is available in Eclipse MDT project [Eclipse, 2009b] as an API. This library is used to examine, create or modify W3C XML Schema. WSGen uses this XSD API to parse the types (simples or complexes) of the operation's messages. The result of the parsing is used to calculate the similarity between two messages (output messages with input messages) of two Web services operations to be composed.

Eclipse BPMN2 Modeler: is an eclipse graphical modeling tool [Eclipse, a] which allows to specify business processes or choreographies using the BPMN 2.0 diagrams. This framework provides a Java library used by programmers to generate BPMN models. The tool is built on Eclipse Graphiti and uses the BPMN 2.0 EMF meta model developed within the Eclipse Model Development Tools (MDT) project. This meta model is compatible with the BPMN 2.0 specification proposed by the Object Management Group. A BPMN model is instantiated by `Bpmn2Factory` class. The created instance is registered by a resource using `Bpmn2ResourceImpl` class. The API provides methods for translating each of the BPMN elements and attributes from EMF objects into DOM objects. DOM objects are serialized to BPMN files.

Apache Tuscany SCA implementation: Apache Tuscany [Apache, 2004b] simplifies the task of developing SOA solutions by providing a comprehensive infrastructure for SOA development and management that is based on Service Component Architecture (SCA) standard. Tuscany SCA Java has implemented the SCA specifications defined by OSOA and OASIS. We used in our tool ArchGen these APIs in order to generate automatically the SCA models from the source code of the service oriented applications. The SCA model is instantiated by the `ScaFactory` interface. The `ScaFactory` interface is also used to instantiate the SCA elements such as: `SCA Composite`, `SCA Component`, `SCA Wire`, `SCA reference`, `SCA Service` and `SCA Promote` which are added into the SCA instance (model). At the end, the SCA instance is registered by a resource using *org.eclipse.emf.ecore.resource.Resource*. The EMF objects are translated into DOM objects, which in turn are serialized to SCA files.

8.5 Summary

In this chapter, a set of tools that implement the proposed ideas in this dissertation are presented. The WSGen tool implements the approach presented in Chapters 5 and 6. It contributes to the generation of Java Web services and BPEL orchestrations throughout the process of migrating of Component based Web applications toward service oriented solutions. In fact, WSGen extracts the Web component archives and provides a specific parser for each of their content (namely, Java classes, Servlets, JSP pages, Web configuration files, HTML files....). And, it generates new Java classes, Java interfaces, WSDL documents, WSDD files, services.xml files, BPEL processes, and their WSDL files. All the generated Java classes are compiled automatically using the JDT API and archived in Axis Archives files. After that, they are deployed by moving them into the Axis server directory. WSGen provides also to the developers an interface to edit, update, save, or choose a set of OCL constraints. In addition, it distributes operations into Web services based on similarity and the coupling metrics.

The ArchGen tool implements the recovering approach presented in Chapter 7. It contributes in the generation of BPMN and SCA architectures starting from the source code of (Web) service oriented systems. The created models can be then opened and visualized directly in a dedicated graphical modeling Framework such as: the Eclipse BPMN2 Modeler and the Eclipse SCA Composite Designer.

In the next chapter, we present the details of the conducted experimentation for evaluating our approaches.

CHAPTER 9

Experimentation: A Case study

What remains eternally incomprehensible in nature is that we can understand.
Albert EINSTEIN.

9.1 Introduction

To evaluate the proposed approaches in this thesis, we conducted two evaluations. First, we evaluated the performance of the proposed process by experimenting it through the migration of three real-world Web applications towards Web service-oriented systems. Second, we evaluated the additional cost induced by the proposed approach by evaluating the developer's effort when (s)he implements extensions to the three Web applications with and without our approach.

9.2 Case study on the migration of Web applications toward Web service oriented solutions

As stated at the beginning of the thesis, when developers want to implement extensions to Web applications, without using our approach, they can either: i) develop these extensions from scratch by writing programs that send HTTP requests to the Web applications and then analyze the returned HTTP responses; or ii) create “manually" Web services that publish the functionality of the Web application[1] and then write programs that invoke these services. We have aforementioned that this task of developing extensions is costly: cumbersome and so time-consuming.

In order to show that our approach reduces the cost of the development of extensions to Web applications, we have conducted an experiment on three real-world Web applications of different sizes. In this experiment, we have in particular addressed the following research questions:

- **RQ1**: What is the performance of the process of identification of operations and Web services that are generated and published?
- **RQ2**: What is the additional cost induced by the proposed approach?

For answering the first research question, we measured in our experiment the performance of our approach taking the definition of “performance” from the information retrieval domain. We measured thus the precision and the recall on the results of the steps in the process: i) the identification of published operations, ii) and the creation of BPEL processes.

For answering the second research question, we have measured: i) the size (in terms of number of statements) of the same extensions developed first without our approach, and then with our approach; ii) the time taken in the development of many extensions to see at what

[1] This task is not really fully manual. We consider the use of tools for annotating code and then generating Web services from this annotated code.

Table 9.1 : Size of the three Web applications

Systems	NSAC	NLOC
E-Auction Application	31	1600
Music Portal Application	32	570
Seekda Application	10	850

time (from how many developed extensions) we can see the benefits of using our approach (initial cost of Web service generation amortized).

The chosen applications for our experiment are: (i) an E-Auction application [Francesca *et al.*,] that provides via its Web interfaces the functionality for buying and selling second-hand goods by bidders and sellers[2]. (ii) An Online Music Portal, which is a JEE Web application[3]. This application offers to users and administrators several functionalities such as: searching, purchasing and managing songs. (iii) A simulated version of a Web service search engine (Seekda). This application provides a set of functionalities, such as, searching for public Web services in the Internet.

Table 9.1 describes the size of the three Web applications, where `NSAC` represents the Number of Server-side scripts And Classes. `NLOC` represents the Number of Lines Of Code in the Web application. These Web applications are considered as inputs for our tool (WSGen: Web Service Generator). We have presented in Section 8.2.2 what are the generated Web services and their compositions obtained from the Seekda Web application.

The experimentation method and all the obtained results have been validated in the publication [Kerdoudi *et al.*, 2016].

9.3 First Experimentation

In order to answer the first research question, we have involved in our experiment four Ph.D. students mastering Java EE. First, each participant is asked to annotate the code of the three Web applications using EJB 3 annotations. Then, we have used Eclipse JEE to generate Web services starting from the annotated classes and methods of these applications. This is what we consider the "manual" Web service generation. We have then measured the number of the generated operations. In addition, we have asked these Ph.D. students to imagine all the possible pertinent operations that can be created from each application's source code. We have calculated the number of these operations. After that, in order to calculate Recall and Precision for the operation identification step, we have measured for each application:

[2]It has been downloaded from the following GitHub repository: https://github.com/FrancescaRodricks/E-Auction-SE-Project

[3]Downloaded from: https://github.com/sahebkanodia/onlinemusicportal

Table 9.2 : Recall and Precision calculation for the operation identification step

Involved developers	Systems	TP	FP	FN	Precision	Recall
Ph.D. Student 1	E-Auction Application	25	9	3	0.73	0.89
	Music Portal Application	10	15	2	0.40	0.83
	Seekda Application	12	2	1	0.85	0.92
Ph.D. Student 2	E-Auction Application	28	9	7	0.75	0.80
	Music Portal Application	12	15	3	0.44	0.80
	Seekda Application	12	2	2	0.85	0.85
Ph.D. Student 3	E-Auction Application	26	9	4	0.74	0.86
	Music Portal Application	13	15	5	0.46	0.72
	Seekda Application	9	2	2	0.81	0.81
Ph.D. Student 4	E-Auction Application	27	9	8	0.75	0.77
	Music Portal Application	14	15	6	0.48	0.70
	Seekda Application	11	2	3	0.84	0.78

Table 9.3 : Recall and Precision calculation for the step of BPEL generation step

Involved developers	Systems	TP	FP	FN	Precision	Recall
Ph.D. Student 1	E-Auction Application	32	4	10	0.88	0.76
	Music Portal Application	10	2	2	0.83	0.83
	Seekda Application	10	2	1	0.83	0.90
Ph.D. Student 2	E-Auction Application	37	4	15	0.90	0.71
	Music Portal Application	11	2	2	0.84	0.84
	Seekda Application	10	2	2	0.83	0.83
Ph.D. Student 3	E-Auction Application	29	4	9	0.87	0.76
	Music Portal Application	13	2	2	0.86	0.86
	Seekda Application	8	2	1	0.80	0.88
Ph.D. Student 4	E-Auction Application	35	4	13	0.89	0.72
	Music Portal Application	15	2	2	0.88	0.88
	Seekda Application	12	2	3	0.85	0.80

- **True Positives (TP)**: the operations identified and published by WSGen and which are also created manually.
- **False Positives (FP)**: the operations created by WSGen, but which are eliminated manually.
- **False Negatives (FN)**: the operations created manually without our approach and which have not been generated by WSGen.
- **True Negatives (TN)**: the operations eliminated by WSGen and are not created manually.

The Precision is the ratio of the number of true positives to the total number of all created operations by WSGen ($TP + FP$). **Precision** = $\frac{TP}{(TP+FP)}$.

The Recall is the ratio of the number of true positives to the number of operations that should be published ($TP + FN$). **Recall** = $\frac{TP}{(TP+FN)}$.

Table 9.2 depicts the obtained values of precision and recall for the three applications. The obtained values show that precision of the operations identification is relatively good except the case of the Music Portal application where the precision is low. In this case, the elimination of the operations provided to the administrator of the application (adding users, ...) cannot effectively be automated and requires the developer knowledge. By this intervention of the developer the value of FP becomes high, which affects negatively the value of the precision.

Besides, in the third column of Table 9.2 (TP) we can see a little difference between the values obtained by the involved developers. This is due to the fact that there are some correct operations which are identified by some developers but they are not identified by the others. This variance is due in general to several factors such as: the level of developer skills, the time spent in analyzing the code (careful code review or not), the complexity and the size of the code and the availability of Web application documentation and architecture.

However, the measures show that the recall rate is relatively high for the three applications. This means that the correctness level of our approach is relatively good (most of the identified operations by WSGen are also created manually). Nevertheless, additional operations are created manually and cannot be identified automatically by WSGen (False Negatives). An example of these operations for the E-Auction application is `getAuctionListForProduct` operation. It returns a list of created auctions for a specific kind of product. This functionality was not provided directly by the E-Auction application, but it was easy to create it by invoking an existing method in the Web application and then filtering its returned results that correspond to a specific product. Additional operations are created manually (False Negatives) by slicing the source code of some programs in the Web applications. The creation of this kind of operations implies complex human thinking and this cannot be fully automated.

There are many variables which are involved in the service and operation identification which made this task too complicated and time-consuming. In such a scenario, identification of candidate services and operations within a large source code is challenging. Various strategies and ways could be adopted such as: using architectural reconstruction approaches [O'Brien *et al.*, 2005], pattern detection [Arcelli *et al.*, 2008] or concept analysis and program slicing techniques [Zhang *et al.*, 2006].

After the evaluation of performance for the operation's identification step, now we have made measures of performance in order to calculate the recall and the precision for the step of the generation of BPEL processes. We have asked the four Ph.D. students to extract all pos-

sible combinations that could be considered as pertinent Web service orchestrations. Then, these have been compared with the generated orchestrations by WSGen. For these measures, the **True Positives** are the number of BPEL processes which are created manually and with WSGen. The **False Positives** are the BPEL processes created by WSGen and eliminated manually. The **False Negatives** are the processes created manually but which have not been generated by WSGen.

As we have seen in the first evaluation, there are some operations which have been created manually and which are not generated by WSGen. As a result, we cannot rely on the previous operation sets to measure recall and precision for BPEL process generation. To deal with this issue, we have decided to consider in our measurement only the operations that are correct (created both by WSGen and manually). After that, we measured the number of BPEL processes that invoke these operations. The obtained values are depicted in Table 9.3. We can see in this table that the precision is high for the three applications, which demonstrates that the step of orchestration generation gives good results with a minimum rate of errors.

Besides, we can observe also in Table 9.3 that the recall is relatively high for the three applications. Despite that, there is a number of orchestrations which are created manually and which are not generated by WSGen. This is due to the use of navigation models for generating BPEL processes. As a result, some combinations of operations cannot be identified from these models, but they are defined manually (False Negatives). An example of these combinations is an orchestration that invokes operations which are created starting from programs in back-end components of Web applications. This kind of operations is not always invoked directly from BPEL processes that are generated by WSGen. This fact does not have an influence on the correctness of our approach, because most of the orchestrations which represent the functionality used by end-users via Web interfaces are created by WSGen.

9.4 Second Experimentation

To answer RQ2, we estimated the developer's effort when (s)he implements extensions to the three Web applications with and without our approach. This part of the experiment is based on the following steps:

- We have implemented extensions to the three Web applications without using our approach and then the same extensions have been developed using our approach. In the extensions, which have been developed without using our approach, we have written Java programs that send HTTP requests to the running Web applications and by analyzing the HTTP responses. The implemented extensions are as follows:
  - For the E-Auction: the extension allows to the user to create alerts by sending an

email when particular products with specific features are offered by the E-Auction application.

  - For the Music Portal: the extension is a mobile front-end for this application, for accessing the searching functionality of the application from an Android tablet.
  - For Seekda: the extension searches for Web service descriptions from the search engine by sending keywords and then retrieves WSDL files from the obtained results (in order to exploit them to make classifications of Web services [Azmeh *et al.*, 2011]).

- We evaluated the number of statements in each implemented extension to each Web application and we compare the obtained values for the two approaches. NS_{WSGen} is the number of statements for implementing an extension by invoking the generated Web services. NS_{HTTP} is the number of statements for implementing an extension by sending HTTP requests to the Web application and analyzing the returned responses.
- We calculated $StmRatio$ which is the ratio between NS_{WSGen} and NS_{HTTP} for the three Web applications ($StmRatio = NS_{HTTP} \ / \ NS_{WSGen}$).

The values for $StmRatio$ for the three applications are: 3.76 for the E-Auction application, 3.6 for the Music Portal application, and 3.2 for Seekda. All the values confirm that the extensions developed using our approach are more than three times smaller than the same extensions developed using HTTP requests (answer to RQ2).

We can observe that the comparison made, between the two ways of developing extensions, to answer the second research question does not take into consideration the initial overhead of our approach. This overhead can be quantified by measuring the number of OCL expressions and the number of interactions with WSGen GUI (number of clicks in order to validate operations, for example). But this cannot be added up to the number of statements and then be compared with the number of statements when developing the extensions without our approach. Then, we have decided to compare the total time spent during the development of several extensions to one of the three Web applications, which is the Music Portal application. In addition to the firstly created extension we have implemented three other simple extensions to this Web application. Then, in order to not limit ourselves to four extensions of this Web application, we have decided to create several fictive (simulated) extensions. For each simulated extension, we vary randomly the value of NS_{HTTP} and we calculate NS_{WSGen} which is equal to: $NS_{HTTP} \times StmRatio$.

The comparison now is based on the calculation of the following values:

- T: represents the total time for generating primitive and composite Web services from the Web application using our approach. This time is the sum of the global time for ex-

Table 9.4 : Time for the four extensions of the Music Portal application

Extensions	T_{HTTP}	T_{WS}
Extension 1	1.94	1.9
Extension 2	1.2	0.85
Extension 3	0.94	0.77
Extension 4	0.92	0.75

ecuting our tool and the time for the intervention of the developer: writing OCL constraints and validating the generated operations and services.

- T_{WSG}: represents the time spent during the implementation of an extension to a Web application by invoking the generated Web services. This time is the sum of the global time needed for a developer to understand the service descriptions and/or the BPEL processes in addition to the time for programming and testing an extension.

- T_{HTTPG}: represents the needed time to implement the same extension without using our approach. This time is the sum of the estimated time for navigating between the interfaces of the Web application in order to learn and save, before the development of the extension, the content of the HTTP requests (and their responses) sent to (respectively, received from) the Web application and the global time to implement and test this extension.

- T_{WS}: is the average time for implementing one statement within an extension using our approach. This time is equal to T_{WSG}/NS_{WSGen}.

- T_{HTTP}: is the average time for implementing one statement within an extension without using our approach. This time is equal to T_{HTTPG}/NS_{HTTP}.

Table 9.4 shows the average values of time for implementing one statement in the four extensions of the Music Portal Web application. We can observe that the values of T_{HTTP} and T_{WS} decrease from one extension to the next one. This is related to the fact that the developer acquires a programming experience during the implementation of an extension, which allows her(him) to implement the next extension in less time. This experience is affected by several factors such as the use of the same API. In this experimentation, we have supposed that all the simulated extensions are implemented using the same API and in the same programming language. Besides, we have estimated the time for rewriting the code of the four extensions (as if it was a "mechanical" task that does not imply thinking). The estimated value of T_{HTTP} and T_{WS} is equal to 0.37 min. This is the minimal time. Now, we would like to predict the values of T_{HTTP} and T_{WS} for each simulated extension. For this, starting from the values in Table 9.4 we have used a regression technique to obtain two curves of trends and their equations. These two equations allowed us to extrapolate the values of T_{HTTP} and T_{WS} for the simulated extensions.

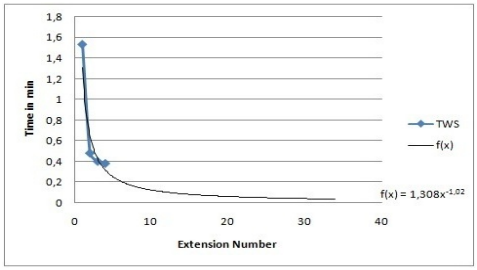

Figure 9.1 : Result of the application of the regression technique for our approach.

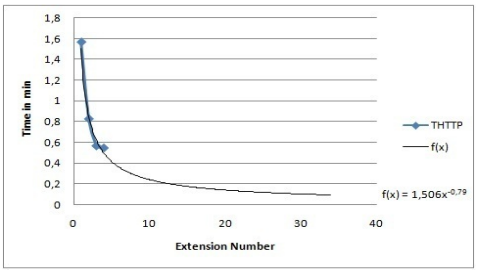

Figure 9.2 : Result of the application of the regression technique for the HTTP approach.

As we have aforementioned, the values in Table 9.4 have a decreasing trend that tends to the value 0.37. So, to determine the curves of trends we used an inverse power regression model, which is defined mathematically as following: $f(x) = a \times x^b$, where a and b are constants, and b is negative. $f(x)$ represents the value of T_{HTTP} or T_{WS} for the extension number: x. It is clear that this function converges to zero. But, we have mentioned that the minimal value of time for T_{HTTP} and T_{WS} is 0.37. Thus, the function that we need in our calculation could be defined as follows: $g(x) = f(x) + 0.37$. To calculate f(x), we have first subtracted the value 0.37 from the values in Table 9.4. After that, we have used these new values as input when we have applied the regression technique. The obtained curves of trends and their equations[4] are shown in Fig 9.1 and Fig 9.2. Now, in order to estimate the global time(T_{HTTPG} and T_{WSG}) for implementing each simulated extension, we use a function h which is defined mathematically as follows: $h(x,r) = g(x) \times r$, where r represents the number of statements NS_{HTTP}[5] or NS_{WSGen}[6] in each simulated extension. Thus, the result of the calculation of time for implementing all the extensions of the Music Portal are shown in Fig9.3.

We have accumulated the global time for each simulated extension (the time to develop the whole extension, not the time for writing one statement). We can observe in the curves that the time for implementing the first extension using our approach is relatively high comparatively to the time spent during the implementation of the same extension without our approach. This

[4] These curves and equations are generated using the tool provided by the Microsoft Office Excel 2007.
[5] NS_{HTTP} is generated randomly.
[6] NS_{WSGen} is calculated by: $NS_{HTTP} \times StmRatio$.

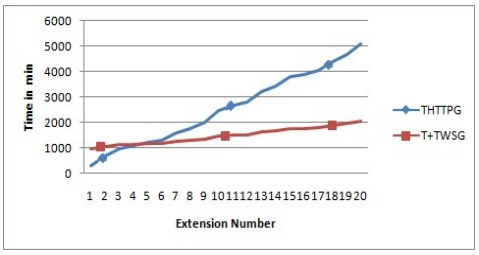

Figure 9.3 : Variation of the time over the number of extensions in the Music Portal Application

time is equal to $T + T_{WSG}$, which represents the initial overhead of our approach (which is considered only one time) and the time for implementing the first extension. We can see in Fig. 9.3, starting from the fourth extension (it represents the point where the two curves intersect), that our method becomes beneficial and the global time for implementing and testing extensions becomes less than the global time for doing the same thing without our approach. In addition, we can observe that the slopes of the two curves are different. The curve related to the development of extensions without our approach grows faster. This shows the amortization of the initial overhead of our approach over the development of multiple extensions (answer to the second part of RQ2).

9.5 Discussion and Threats To Validity

For the external validity, and in order to generalize the experiment results of the case study, we need to take into consideration several aspects such as the type and the size of the selected Web applications. We conducted our experiment on different types of Web applications that have relatively medium sizes. To achieve results which can be confidently generalized, we need to run this experiment on larger Web applications developed using different technologies (languages and/or frameworks). But the fact that our applications have medium sizes helped us in manually creating Web services in a reasonable time. In addition, the hypothesis about the use of HTTP requests (using java.net API of the standard library of the JDK) for developing extensions slightly biases the experiment. Indeed, this is not the most effective way of doing, but we have considered it because it is the most straightforward and the standard way for developing Java-based extensions of Web applications (for Web scraping, for instance). Besides, this is the only way of developing such extensions for Web applications that do not provide a service-oriented REST (or SOAP-based) API, which is the case of most Web applications today.

On what concerns the internal validity, we can be tempted to say that, as we have been involved (not in the manual annotation of the Web application code to generate Web services in the first part of the experiment, but) in the development of the extensions without WSGen in the second part of the experiment, the results would be biased. This fact does not have an influence on the experiment results. Indeed, the way of developing the extensions by using

HTTP requests is greatly different from using Web services. The fact that the developers know what would be the generated Web service interfaces of the Web applications does not impact their way of programming the HTTP requests using the java.net API.

9.6 Summary

In this chapter, we have presented the setup, the process and the result of two evaluations. Basically, they aim at proving the applicability of the proposed ideas and measuring the performance of the proposed approach. The obtained results of experiments, showed the applicability and the cost-effectiveness of the approach. We alleviate in particular the developer from the complexity of creating these Web services (individual and composite) manually. As we have explained above, this task is particularly challenging, time consuming and error prone, especially, when we parse a complex and large Web application, and when we create Web services starting from Web interfaces, which are implemented with a mixing of different paradigms such as, HTML, CSS, JavaScript and Java.

In addition, we have shown that in our method, the global time for implementing and testing extensions is much less than the global time for doing the same thing without our approach. Indeed, time related to the development of extensions without our approach grows faster every time we implement new extensions, while our approach, starting from the fourth extension, becomes beneficial.

In the near future, we plan to evaluate the accuracy, the performance of our software recovering approach on case study in which we apply our techniques to produce service architectures of one of the open source implementations of OSGi Framework such as Eclipse Equinox.

CHAPTER 10

Conclusion and Future Work

10.1 Summary

THE work presented in this thesis contributes in opening Web applications for third party development. Nowadays, there is a real need for shifting from Web applications targeting exclusively humans into Web service-oriented applications. Examples of scenarios where this need is felt, for example, (i) when we want to build mobile applications by using data from existing Web applications, or (ii) when we want to implement a new "niche" business logic in the Web, underlying a large/famous Web application. After this shift, the obtained result will thus enable third tier developers to build systems by reusing services provided by the existing Web applications instead of creating them from scratch or dealing with complex HTTP interactions. In this thesis, we have addressed the problem of opening Component-Based Web Applications for third party development. Web components are seen here as software artifacts embedding business logic code and exporting Web interfaces. We proposed an approach (in Chapter 5) that helps developers to create Web service-oriented systems starting from the Web user interfaces of these applications. We considered in our approach these deployed artifacts (embedding Web interfaces) as remote APIs (as Web Services) that offer the opportunity for developers to extend the functionality provided by these services and exploit the resources used by them. Indeed, the third party developers use the WSDL descriptions of these Web services to create their client applications. These latter can send XML-based messages (SOAP) to the chosen Web service. The same kind of messages are returned back to these client applications, containing answers to their requests. Upon these results, more actions can be performed in

order to implement some new business-logic.

In addition to the created individual Web services from Web interfaces and existing primitive functionality, we proposed in Chapter 6 a method to generate composite Web services by assembling the created individual Web services in order to provide coarse-grained functionalities. The approach relies on the creation of Web service orchestrations and choreographies. Both concepts imply coordination or control the act of making individuals Web services work together to form some consistent overall process. Orchestration refers to coordination at the level of a single participant's process, whereas choreography refers to the global view, spanning multiple participants. The orchestrations in our approach are expressed using BPEL, which is one of the leading languages in this field. The choreographies are generated starting from the method invocations located in the source code of the generated individual Web services. We have proposed the necessary algorithms and techniques to generate BPEL processes starting from Web user interfaces navigations and to generate the Web service choreographies as a set of Web service requests embedded in the source code. All these "emerging Web services" contribute in opening Web applications for third-tier extension development.

In order to present more accurately the processing performed in Web application to create service oriented solutions, we have proposed (in Chapter 4) a formal model that represents in an unambiguous way both kinds of systems as graphs. These mathematical notations are used to present the migration as a mapping between graphs. This formal definition helps in the general understanding, and it gives a solid reference point for the rest of the thesis.

Besides, software evolution is an inescapable activity in the software lifecycle [Lehman et Belady, 1985]. In this activity, system comprehension plays an important role in understanding the overall structure and behavior of a software before starting to apply changes to it. This is particularly critical for new developers in the software project team or if evolution is performed a long time after the initial development of this software. We proposed (in Chapter 7) an approach for recovering service oriented architectures from the source code of a (Web) service oriented system. These architectures offer to developers (third party developers or maintenance's engineers) a support model of a high level of abstraction (in contrast to source code) which can help them in understanding their systems. We provide two categories of support models (of a high level of abstraction), which are: (i) BPMN models which represent the behavior of a service composition, and (ii) SCA specifications which represent the structure of the service composition as a set of components connected with contractually specified interfaces. The interfaces are the provided and the required services from these components. The architecture recovering is not performed in an ad-hoc way, but through explicit transformation rules, which are extensible, easily testable and quickly debuggable.

All the proposed ideas in this thesis are implemented thought two tools called: WSGen and ArchGen. The WSGen allows to generate primitive and composite Web services from a

Component-based Web application and the ArchGen allows to generate BPMN models and SCA specifications from the source code of (Web) Service Oriented Systems. The details about the implemented tools are presented in Chapter 8.

Finally, we have conducted two experiments to evaluate the proposed approaches (in Chapter 9). In the first experimentation, we measured the performance of our approach by migration of three real world Web applications into composite and primitive Web services. Comparing to the manual creation of these Web services, the obtained results, have shown a good values of precision and recall. In the second experimentation, we investigated the additional cost induced by the proposed approach. The obtained experiment results confirm that the extensions developed using our approach are more than three times smaller than the same extensions developed using HTTP requests. Besides, we evaluated the architecture recovery approach on set of examples as a part of the evaluation process. The developers that were involved in this evaluation confirmed that the generated models are very helpful. Experimenting in depth this step of our approach on large applications with real-world developers is one of the perspectives of this work. However, it is commonly agreed that recovering high-level architecture descriptions is helpful during software comprehension.

To sum up, we list the potential beneficiaries of the proposed ideas in this thesis: i) the organization which holds the rights on the Web application and whose developers would use the proposed method: migrating a Web application of this organization towards a service-oriented one enables the organization to modernize its “patrimony” and to shift to a new paradigm (of service orientation); ii) third party developers: they will be able to develop new business processes, potentially with financial benefits, starting from the generated Web services. The development of this “ecosystem” (composed of third party developers) around the generated services should necessarily bring a return on investment for the organization holding the rights on the original Web application.

10.2 Perspectives

In the near future, we plan to extend the proposed method by implementing 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 QVT-compliant language [OMG., 2008] and thus integrate our solution in a Model-Driven Engineering process.

Furthermore, we plan to address more accurately the security issues when migrating Web application toward Web service oriented systems. In fact, in the current work, we deal with Web interfaces that use a secure protocol such as TSL (SSL). Actually, Web applications start using a third party secure delegation service (such as OAuth authorization [Leiba, 2012]) to

enhance their access security. As a future work, we intend to study the migration of such kind of Web applications towards Web service-oriented systems that use this secure delegation service. For example, the implementation of the scenario of delegation provided by OAuth could be migrated into a Web service orchestration where the user is involved to introduce the scope of authorization.

The results of the experiments demonstrated that service identification is challenging, and there are several strategies and ways that could be adopted such as using architectural reconstruction approaches [O'Brien *et al.*, 2005], pattern detection [Arcelli *et al.*, 2008] or concept analysis and program slicing techniques [Zhang *et al.*, 2006]. Indeed, we plan in the future studying these strategies in order to improve this step of our approach so that obtaining more precise results.

Future work includes also the use of a hierarchical clustering technique like Weighted Combined Algorithm [Maqbool et Babri, 2004] in service architecture recovery approach. We illustrated in Chapter 7 the grouping the components in a composite is based on their category and developer knowledge. We plan in future to use the Weighted Combined Algorithm, where, we represent each component by a cluster, and we compute the pair-wise similarity between all the clusters and then we combine the two most similar clusters into a new cluster. This is repeated until all the elements will be clustered or the desired number of clusters is achieved.

In our thesis, the recovered architectures are created based on static analysis of the service oriented applications. In the long term, we would like to analyze dynamically the artifacts, like execution traces to identify at runtime dynamic service dependencies. In this kind of systems, an optimized architectures can be recovered at runtime, instead of dealing with static "spaghetti" architectures. This optimization is possible because architecture elements are not all involved in the running system. Thereby, these elements can be hidden in the dynamic architecture description, and concretely running elements can be highlighted. By simplifying architecture descriptions, they enable developers to make like a quick "inventory" of what is concretely running, among all what composes their system, at a particular execution time (bug occurrence, for example). They can easily identify which component is consuming a particular failing service, for instance.

List of Figures

List of Tables

List of Listings

Bibliography

[Alliance, 2014] OSGi Alliance. Osgi core specification release 6, 2014.

[Allier *et al.*, 2010] Simon Allier, Houari A Sahraoui, Salah Sadou, et Stéphane Vaucher. Restructuring object-oriented applications into component-oriented applications by using consistency with execution traces. In *Proceedings of the 13th International Conference on Component-Based Software Engineering (CBSE)*, pages 216–231. Springer, 2010.

[Almonaies *et al.*, 2010] Asil A Almonaies, James R Cordy, et Thomas R Dean. Legacy system evolution towards service-oriented architecture. In *International Workshop on SOA Migration and Evolution*, pages 53–62, 2010.

[Almonaies *et al.*, 2013] Asil A. Almonaies, Manar H. Alalfi, James R. Cordy, et Thomas R. Dean. A Framework for Migrating Web Applications to Web Services. In *Proc. of ICWE*, 2013.

[Ameller *et al.*, 2015] David Ameller, Xavier Burgués, Oriol Collell, Dolors Costal, Xavier Franch, et Mike P Papazoglou. Development of service-oriented architectures using model-driven development: A mapping study. *Information and Software Technology*, 62:42–66, 2015.

[Andritsos et Tzerpos, 2005] Periklis Andritsos et Vassilios Tzerpos. Information-theoretic software clustering. *IEEE Trans. Softw. Eng.*, 31(2):150–165, 2005.

[Anquetil *et al.*, 2009] Nicolas Anquetil, Jean-Claude Royer, Pascal Andre, Gilles Ardourel, Petr Hnetynka, Tomas Poch, Dragos Petrascu, et Vladiela Petrascu. Javacompext: Extracting architectural elements from java source code. In *Proc. of WCRE'09*. IEEE, 2009.

[Apache, 1999] Software Foundation Apache. The Apache Xerces Project. http://xerces.apache.org/, 1999.

[Apache, 2003] Software Foundation Apache. JaxMe Java Source Framework. http://www.java2s.com/Code/Jar/j/Downloadjaxmejs02jar.htm, 2003.

[Apache, 2004a] Software Foundation Apache. Apache Axis, Web Service Project. http://axis.apache.org/axis/, 2004.

[Apache, 2004b] Software Foundation Apache. Apache Tuscany. http://tuscany.apache.org/home.html, 2004.

[Apache, 2005] Software Foundation Apache. Tomcat Apache Project. http://tomcat.apache.org/, 2005.

[Apache., 2013] Apache. Apache Orchestration Director Engine. http://ode.apache.org/, 2013.

[Arcelli *et al.*, 2008] Francesca Arcelli, Christian Tosi, et Marco Zanoni. Can design pattern detection be useful for legacy systemmigration towards soa. In *Proceedings of the 2nd IEEE international ICSE workshop on Systems development in SOA environments (SDSOA '08)*, pages 63–68. ACM, 2008.

[Aït-Bachir, 2008] Ali Aït-Bachir. Measuring similarity of service interfaces. In *ICSOC PhD Symposium*, 2008.

[Azmeh *et al.*, 2011] Z. Azmeh, M. Driss, F. Hamoui, M. Huchard, Moha N., et C. Tibermacine. Selection of composable web services driven by user requirements. In *Proc. of IEEE ICWS*, 2011.

[Badri et Badri, 2004] Linda Badri et Mourad Badri. A proposal of a new class cohesion criterion: An empirical study. *Journal of Object Technology*, 3(4):145–159, 2004.

[Bauer et Huget, 2004] Bernhard Bauer et Marc-Philippe Huget. Modelling web service composition with uml 2.0. *International journal of Web engineering and technology*, 1(4):484–501, 2004.

[Bauer et Muller, 2004] Bernhard Bauer et Jorg P. Muller. Mda applied: From sequence diagrams to web service choreography. In *Proc. of ICWE*, 2004.

[Baumgartner *et al.*, 2001] Robert Baumgartner, Sergio Flesca, et Georg Gottlob. Visual web information extraction with lixto. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 119–128, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[Baumgartner *et al.*, 2004] Robert Baumgartner, Georg Gottlob, Marcus Herzog, et Wolfgang Slany. Interactively adding web service interfaces to existing web applications. In *Proceedings of the International Symposium on Applications and the Internet (SAINT),*, pages 74–80. IEEE Computer Society, 2004.

[Beisiegel *et al.*, 2009] Michael Beisiegel, Khanderao Khand, Anish Karmarkar, Sanjay Patil, Michael Rowley, Martin Chapman, et Mike Edwards. Service Component Architecture Assembly Model Specification Version 1.1. http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd03.html, 2009.

[Belushi et Baghdadi, 2007] Wesal A. Belushi et Youcef Baghdadi. An Approach to Wrap Legacy Applications into Web Services. In *Proceedings of International Conference on Service Systems and Service Management*, pages 1–6. IEEE Computer Society, 2007.

[Benatallah *et al.*, 2003] Boualem Benatallah, Marlon Dumas, Marie-Christine Fauvet, et Fethi A. Rabhi. Patterns and skeletons for parallel and distributed computing. chapitre Towards Patterns of Web Services Composition, pages 265–296. Springer-Verlag, 2003.

[Bennett, 1996] Keith Bennett. Software evolution: past, present and future. *Information and software technology*, 38(11):673–680, 1996.

[Bisbal *et al.*, 1999] Jesús Bisbal, Deirdre Lawless, Bing Wu, et Jane Grimson. Legacy information systems: Issues and directions. *IEEE software*, 16(5):103, 1999.

[Boustil *et al.*, 2014] Amel Boustil, Ramdane Maamri, et Zaidi Sahnoun. A semantic selection approach for composite web services using owl-dl and rules. *Service Oriented Computing and Applications*, 8(3):221–238, 2014.

[Box *et al.*, 2000] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, et Dave Winer. Simple object access protocol (soap) 1.1. https://www.w3.org/TR/2000/NOTE-SOAP-20000508/, 2000.

[Bray *et al.*, 1998] Tim Bray, Jean Paoli, C Michael Sperberg-McQueen, Eve Maler, et François Yergeau. Extensible markup language (xml), 1998.

[Briand *et al.*, 1996] Lionel C. Briand, Sandro Morasca, et Victor R. Basili. Property-based software engineering measurement. *IEEE TSE*, 22(1):68–86, 1996.

[Briand *et al.*, 1998] Lionel C. Briand, John W. Daly, et Jürgen Wüst. A unified framework for cohesion measurement in object-oriented systems. *Empirical Software Engineering*, 3(1):65–117, 1998.

[Briand *et al.*, 2005] L.C. Briand, Y. Labiche, M. Di Penta, et H. Yan-Bondoc. An experimental investigation of formality in uml-based development. *IEEE TSE*, 31:833–849, 2005.

[Brown *et al.*, 2005] Alan W Brown, Jim Conallen, et Dave Tropeano. Introduction: Models, modeling, and model-driven architecture (mda). In *Model-Driven Software Development*, pages 1–16. Springer, 2005.

[Burns et Kitain, 2009] Ed Burns et Roger Kitain. Javaserver™ faces 2.0 final release specification. http://download.oracle.com/otndocs/jcp/jsf-2.0-fr-full-oth-JSpec/, 2009.

[Canfora *et al.*, 2008] Gerardo Canfora, Anna Rita Fasolino, Gianni Frattolillo, et Porfirio Tramontana. A wrapping approach for migrating legacy system interactive functionalities to service oriented architectures. *JSS*, 81(4):463–480, 2008.

[Chardigny *et al.*, 2008] Sylvain Chardigny, Abdelhak Seriai, Mourad Oussalah, et Dalila Tamzalit. Extraction of component-based architecture from object-oriented systems. In *In proceeding of the Seventh IEEE/IFIP Working Conference on Software Architecture (WICSA)*, pages 285–288. IEEE, 2008.

[Chinnici *et al.*, 2007] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, et Sanjiva Weerawarana. Web services description language (wsdl) version 2.0 part 1: Core language. http://www.w3.org/TR/2007/REC-wsdl20-20070626/, 2007.

[Clement *et al.*, 2004] Luc Clement, Andrew Hately, Claus von Riegen, Tony Rogers, et al. UDDI Version 3.0.2, UDDI Spec Technical Committee Draft, 2004.

[Corazza *et al.*, 2010] Anna Corazza, Sergio Di Martino, et Giuseppe Scanniello. A probabilistic based approach towards software system clustering. In *In proceedings of the 14th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 88–96. IEEE, 2010.

[Crasso *et al.*, 2008] Marco Crasso, Alejandro Zunino, et Marcelo Campo. Query by example for web services. In *Proc. of ACM SAC*, 2008.

[Daigneau, 2011] Robert Daigneau. *Service design patterns: fundamental design solutions for SOAP/WSDL and RESTful web services.* Addison-Wesley, 2011.

[Decker *et al.*, 2008] Gero Decker, Oliver Kopp, Frank Leymann, Kerstin Pfitzner, et Mathias Weske. Modeling service choreographies using bpmn and bpel4chor. In *Proc. of CAiSE*, 2008.

[Demange *et al.*, 2013] Anthony Demange, Naouel Moha, et Guy Tremblay. Detection of soa patterns. In *In Proceedings of the 11th International Conference on Service-Oriented Computing ICSOC'13*, pages 114–130. Springer, 2013.

[Derrick *et al.*, 2002] Oswald Derrick, Raha Somik, Ian Macfarlane, et David Walters. HTML Parser website. http://htmlparser.sourceforge.net, 2002.

[Dierks et Rescorla, 2006] T. Dierks et E. Rescorla. The transport layer security (tls) protocol. In *IETF RFC 4346*, 2006.

[Djelloul *et al.*, 2009] Bouchiha Djelloul, Malki Mimoun, et Mostefai Abd El Kader. Towards reengineering web applications to web services. *The International Arab Journal of Information Technology (IAJIT)*, 6(4), 2009.

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

[Ducasse et Pollet, 2009] Stéphane Ducasse et Damien Pollet. Software architecture reconstruction: A process-oriented taxonomy. *IEEE Transactions on Software Engineering*, 35(4):573–591, 2009.

[Dumez *et al.*, 2008] Christophe Dumez, Jaafar Gaber, et Maxime Wack. Model-driven engineering of composite web services using uml-s. In *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, pages 395–398. ACM, 2008.

[Eclipse, a] Foundation Eclipse. BPMN2 Modeler website. https://www.eclipse.org/bpmn2-modeler/.

[Eclipse, b] Foundation Eclipse. SCA Tools. https://eclipse.org/soa/sca/.

[Eclipse, 2001] Foundation Eclipse. Eclipse Java development tools (JDT) Project. http://www.eclipse.org/jdt/, 2001.

[Eclipse, 2005] Foundation Eclipse. Eclipse BPEL Model Tool. http://www.eclipse.org/bpel/developers/model.php, 2005.

[Eclipse, 2009a] Foundation Eclipse. Eclipse Modeling Framework Project. http://www.eclipse.org/modeling/emf/?project=emf, 2009.

[Eclipse, 2009b] Foundation Eclipse. Model Development Tools website. http://www.eclipse.org/modeling/mdt/, 2009.

[Erl, 2008] Thomas Erl. *SOA: Principles of Service Design*. Prentice Hall, 2008.

[Erl, 2009] Thomas Erl. *SOA Design Patterns*. Prentice Hall, 2009.

[Etzkorn *et al.*, 2004] Letha H. Etzkorn, Sampson E. Gholston, Julie L. Fortune, Cara E. Stein, Dawn Utley, Phillip A. Farrington, et Glenn W. Cox. A comparison of cohesion metrics for object-oriented systems. *Information and Software Technology*, 46(10):677–687, 2004.

[Falkner et Jones, 2004] Jayson Falkner et Kevin Jones. *Servlets and JavaServer Pages: The J2EE Technology Web Tier*. Addison-Wesley, 2004.

[Fei et Wang, 2004] Yui-Ku Fei et Zhijian Wang. A concept model of web components. In *Proc. of IEEE SCC*, 2004.

[Fielding, 2000] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[Finkelstein et Kramer, 2000] Anthony Finkelstein et Jeff Kramer. Software engineering: a roadmap. In *Proc. of ICSE*, 2000.

[Flanagan, 2011] David Flanagan. *JavaScript - The Definitive Guide (6th ed.)*. O'Reilly, 2011.

[Forster *et al.*, 2013] Thomas Forster, Thorsten Keuler, Jens Knodel, et Michael-Christian Becker. Recovering component dependencies hidden by frameworks–experiences from analyzing osgi and qt. In *Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering*, CSMR '13, pages 295–304. IEEE Computer Society, 2013.

[Francesca *et al.*,] Rodricks Francesca, Chauhan Sunil, Pascoala D'Souza, Kumar Subodh, et Fernandes Leanne. E-Auction System, project of Goa University . https://github.com/FrancescaRodricks/E-Auction-SE-Project.

[Franciscus et McClanahan, 2002] George Franciscus et Craig R McClanahan. *Struts in Action: Building web applications with the leading Java framework*. Manning Publications Co., 2002.

[Fuhr *et al.*, 2013] Andreas Fuhr, Tassilo Horn, Volker Riediger, et Andreas Winter. Model-driven software migration into service-oriented architectures. *Computer Science-Research and Development*, 28(1):65–84, 2013.

[Garcia *et al.*, 2011] Joshua Garcia, Daniel Popescu, Chris Mattmann, Nenad Medvidovic, et Yuanfang Cai. Enhancing architectural recovery using concerns. In *In proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 552–555. IEEE Computer Society, 2011.

[Garcia *et al.*, 2013a] Joshua Garcia, Igor Ivkovic, et Nenad Medvidovic. A comparative analysis of software architecture recovery techniques. In *Proceedings of the IEEE/ACM 28th International Conference on Automated Software Engineering (ASE 2013)*, pages 486–496. IEEE, 2013.

[Garcia *et al.*, 2013b] Joshua Garcia, Ivo Krka, Chris Mattmann, et Nenad Medvidovic. Obtaining ground-truth software architectures. In *Proceedings of the 35th International Conference on Software Engineering (ICSE 2013)*, pages 901–910. IEEE Press, 2013.

[Geary et Horstmann, 2004] David Geary et Cay Horstmann. *Core JavaServer Faces,*. Addison Wesley, 2004.

[Gronmo *et al.*, 2004] R. Gronmo, D. Skogan, I. Solheim, et J. Oldevik. Model-driven web service development. *IJWSR*, 1(4):1–13, 2004.

[Guo *et al.*, 2005] He Guo, Chunyan Guo, Feng Chen, et Hongji Yang. Wrapping client-server application to web services for internet computing. In *Proceedings of the Sixth International Conference on Parallel and Distributed Computing Applications and Technologies*, PDCAT '05, pages 366–370, Washington, DC, USA, 2005. IEEE Computer Society.

[Hadley, 2009] Marc Hadley. Web application description language (wadl). https://www.w3.org/Submission/wadl/, 2009.

[Haesen *et al.*, 2008] Raf Haesen, Monique Snoeck, Wilfried Lemahieu, et Stephan Poelmans. On the definition of service granularity and its architectural impact. In *Advanced Information Systems Engineering*, éditeurs Zohra Bellahsène et Michel Léonard, volume 5074 de *Lecture Notes in Computer Science*, pages 375–389. Springer Berlin Heidelberg, 2008.

[Hall *et al.*, 2011] Richard Hall, Karl Pauls, Stuart McCulloch, et David Savage. *OSGi in action: Creating modular applications in Java*. Manning Publications Co., 2011.

[Han et Tokuda, 2008] Hao Han et Takehiro Tokuda. Wike: A web information/knowledge extraction system for web service generation. In *Proc. of ICWE*, 2008.

[Havey, 2005] Michael Havey. *Essential business process modeling*. " O'Reilly Media, Inc.", 2005.

[Herzog et Gottlob, 2001] Marcus Herzog et Georg Gottlob. Infopipes: a flexible framework for m-commerce applications. In *Technologies for E-Services*, pages 175–186. Springer, 2001.

[Holovaty et Kaplan-Moss, 2009] Adrian Holovaty et Jacob Kaplan-Moss. *The definitive guide to Django: Web development done right*. Apress, 2009.

[Immonen et Pakkala, 2014] Anne Immonen et Daniel Pakkala. A survey of methods and approaches for reliable dynamic service compositions. *Service Oriented Computing and Applications*, 8(2):129–158, 2014.

[Jendrock *et al.*, 2014] Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Kim Haase, et William Markito. *The Java EE 7 Tutorial: Getting Started with Web Applications*. Addison-Wesley Professional, 2014.

[Jiang et Stroulia, 2004] Yingtao Jiang et Eleni Stroulia. Towards reengineering web sites to web-services providers. In *Proceedings of the Eighth European Conference on Software Maintenance and Reengineering, (CSMR'04)*, pages 296–305. IEEE, 2004.

[Johnston et Brown, 2006] Simon K. Johnston et Alan W. Brown. A model-driven development approach to creating service-oriented solutions. In *Proc. of ICSOC*, 2006.

[Josuttis, 2007] Nicolai M Josuttis. *SOA in practice: the art of distributed system design*. " O'Reilly Media, Inc.", 2007.

[Kebir *et al.*, 2012] Selim Kebir, Abdelhak-Djamel Seriai, Sylvain Chardigny, et Allaoua Chaoui. Quality-centric approach for software component identification from object-oriented code. In *In proceeding of the IEEE/IFIP WICSA and ECSA*, pages 181–190. IEEE, 2012.

[Kerdoudi *et al.*, 2016] Mohamed Lamine Kerdoudi, Chouki Tibermacine, et Salah Sadou. Opening web applications for third-party development: a service-oriented solution. *Service Oriented Computing and Applications*, pages 1–27, 2016.

[Khadka *et al.*, 2013] Ravi Khadka, Amir Saeidi, Andrei Idu, Jurriaan Hage, et Slinger Jansen. Legacy to soa evolution: A systematic literature review. In *A. D. Ionita, M. Litoiu, G. Lewis (Eds.) Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments*. IGI Global, 2013.

[Kokash, 2006] Natallia Kokash. A comparison of web service interface similarity measures. In *Proc. of the Third Starting AI Researchers' Symposium*, 2006.

[Kulkarni et Dwivedi, 2008] Naveen Kulkarni et Vishal Dwivedi. The role of service granularity in a successful soa realization a case study. In *Proceedings of the 2008 IEEE Congress on Services - Part I*, SERVICES '08, 2008.

[Lee *et al.*, 2005] Roger Y. Lee, Ashok K. Harikumar, Chia-Chu Chiang, Hae Sool Yang, Haeng-Kon Kim, et Byeongdo Kang. A framework for dynamically converting components to web services. In *Proc. of SERA*, 2005.

[Lehman et Belady, 1985] M.M. Lehman et L. Belady. *Program Evolution: Process of Software Change*. London: Academic Press, 1985.

[Leiba, 2012] Barry Leiba. Oauth web authorization protocol. *IEEE Internet Computing*, 16(1):74–77, 2012.

[Lewis *et al.*, 2006] Grace Lewis, Edwin J. Morris, et Dennis Smith. Analyzing the reuse potential of migrating legacy components to a service-oriented architecture. In *Proc. of CSMR*, 2006.

[Liang *et al.*, 2006] Qianhui Althea Liang, Jen-Yao Chung, Steven Miller, et Yang Ouyang. Service pattern discovery of web service mining in web service registry-repository. In *In Proceedings of the IEEE International Conference on e-Business Engineering (ICEBE'06)*, pages 286–293. IEEE, 2006.

[Lorenzo *et al.*, 2007] Giusy Di Lorenzo, Anna Rita Fasolino, Lorenzo Melcarne, Porfirio Tramontana, et Valeria Vittorini. Turning web applications into web services by wrapping techniques. In *Proc. of the 14th Working Conference on Reverse Engineering (WCRE)*, pages 199–208. IEEE Computer Society, 2007.

[Mancoridis *et al.*, 1999] Spiros Mancoridis, Brian S Mitchell, Yihfarn Chen, et Emden R Gansner. Bunch: A clustering tool for the recovery and maintenance of software system structures. In *In Proceedings of the IEEE International Conference on Software Maintenance, (ICSM'99)*, pages 50–59. IEEE, 1999.

[Maqbool et Babri, 2004] Onaiza Maqbool et Haroon Atique Babri. The weighted combined algorithm: A linkage algorithm for software clustering. In *Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. Eighth European Conference on*, pages 15–24. IEEE, 2004.

[Maras *et al.*, 2012] Josip Maras, Jan Carlson, et Ivica Crnkovi. Extracting client-side web application code. In *Proc. of WWW*, 2012.

[Maras *et al.*, 2013] Josip Maras, Maja Stula, Jan Carlson, et Ivica Crnkovic. Identifying code of individual features in client-side web applications. *IEEE TSE*, 39(12):1680–1697, 2013.

[Marinho *et al.*, 2009] Anderson Marinho, Leonardo Gresta Paulino Murta, et Cláudia Werner. Extending a software component repository to provide services. In *Proc. of ICSR*, 2009.

[McAffer *et al.*, 2010] Jeff McAffer, Paul VanderLei, et Simon Archer. *OSGi and Equinox: Creating highly modular Java systems*. Addison-Wesley Professional, 2010.

[Medjahed et Bouguettaya, 2005] Brahim Medjahed et Athman Bouguettaya. A multilevel composability model for semantic web services. *IEEE TKDE*, 17(7):954–968, Juillet 2005.

[Milanovic et Malek, 2004] Nikola Milanovic et Miroslaw Malek. Current solutions for web service composition. *IEEE Internet Comp.*, 8:51–59, 2004.

[OASIS., 2007] OASIS. Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, 2007.

[O'Brien *et al.*, 2005] Liam O'Brien, Dennis Smith, et Grace Lewis. Supporting migration to services using software architecture reconstruction. In *STEP*, pages 81–91. IEEE Computer Society, 2005.

[Oh *et al.*, 2008] Seog-Chan Oh, Dongwon Lee, et Soundar R. T. Kumara. Effective web service composition in diverse and large-scale service networks. *IEEE Trans. Serv. Comput.*, 1(1):15–32, 2008.

[OMG.,] OMG. UML Profile for Enterprise Distributed Object Comp. http://www.omg.org/technology/documents/formal/edoc.htm.

[OMG., 2006] OMG. Object Constraint Language specification, version 2.0, document formal/2006-05-01. http://www.omg.org/spec/OCL/2.0/, 2006.

[OMG., 2008] OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT). http://www.omg.org/spec/QVT/, 2008.

[OMG., 2011a] OMG. Business Process Model and Notation (BPMN) Version 2.0. http://www.omg.org/spec/BPMN/2.0/, 2011.

[OMG., 2011b] OMG. Unified Modeling Language (UML) Superstructure specification, Version 2.4.1. http://www.omg.org/spec/UML/2.4.1/, 2011.

[OMG, 2012] Object Management Group OMG. Specification of the service oriented architecture modeling language (soaml), version 1.0.1. Document formal/2012-05-10. Object Management Group Website: http://www.omg.org/spec/SoaML/1.0.1/, May 2012.

[Oracle,] Oracle. Java platform, enterprise edition: The java ee tutorial. https://docs.oracle.com/javaee/7/tutorial/.

[Paik *et al.*, 2014] Incheon Paik, Wuhui Chen, et Michael N. Huhns. A scalable architecture for automatic service composition. *IEEE Trans. Serv. Comput.*, 7(1):82–95, 2014.

[Perepletchikov *et al.*, 2007] Mikhail Perepletchikov, Caspar Ryan, Keith Frampton, et Heinz W. Schmidt. A formal model of service-oriented design structure. In *Proc. of ASWEC*. IEEE Computer Society, 2007.

[Rao et Su, 2005] Jinghai Rao et Xiaomeng Su. A survey of automated web service composition methods. In *Proceedings of the First International Conference on Semantic Web Services and Web Process Composition*, SWSWPC'04, pages 43–54, Berlin, Heidelberg, 2005. Springer-Verlag.

[Razavian et Lago, 2015] Maryam Razavian et Patricia Lago. A systematic literature review on soa migration. *Journal of Software: Evolution and Process*, 27(5):337–372, 2015.

[Ren *et al.*, 2011] Kaijun Ren, Nong Xiao, et Jinjun Chen. Building quick service query list using wordnet and multiple heterogeneous ontologies toward more realistic service composition. *IEEE T. Services Computing*, 4(3):216–229, 2011.

[Rodriguez *et al.*, 2010] Juan Manuel Rodriguez, Marco Crasso, Alejandro Zunino, et Marcelo Campo. Improving web service descriptions for effective service discovery. *Science of Computer Programming*, 75(11):1001–1021, 2010.

[Segev et Toch, 2009] Aviv Segev et Eran Toch. Context-based matching and ranking of web services for composition. *IEEE Trans. Serv. Comput.*, 2(3):210–222, 2009.

[Seriai *et al.*, 2014a] Abderrahmane Seriai, Salah Sadou, Houari Sahraoui, et Salma Hamza. Deriving component interfaces after a restructuring of a legacy system. In *In proceeding of the IEEE/IFIP WICSA*, pages 31–40. IEEE, 2014.

[Seriai *et al.*, 2014b] Abderrahmane Seriai, Salah Sadou, et Houari A Sahraoui. Enactment of components extracted from an object-oriented application. In *In proceeding of the ECSA*, pages 234–249. Springer, 2014.

[Shaw et Garlan, 1996] M. Shaw et D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[Sindhgatta et Ponnalagu, 2008] Renuka Sindhgatta et Karthikeyan Ponnalagu. Locating components realizing services in existing systems. In *Proceedings of the IEEE International Conference on Services Computing, SCC'08.*, volume 1, pages 127–134. IEEE, 2008.

[Sneed, 2006] Harry M. Sneed. Integrating legacy software into a service oriented architecture. In *Proc. of CSMR*, 2006.

[Sommerville, 2011] Ian Sommerville. *Software Engineering (9th Edition)*. International Computer Science Series. Pearson, 2011.

[Sosa *et al.*, 2013] Encarna Sosa, Pedro J Clemente, Jose M Conejero, et Roberto Rodriguez-Echeverria. A model-driven process to modernize legacy web applications based on service oriented architectures. In *Proc. of the 15th IEEE International Symposium on Web Systems Evolution (WSE)*, pages 61–70. IEEE Computer Society, 2013.

[Staron, 2006] Miroslaw Staron. Adopting model driven software development in industry–a case study at two companies. In *Model Driven Engineering Languages and Systems*, pages 57–72. Springer, 2006.

[Szyperski *et al.*, 1999] Clemens Szyperski, Jan Bosch, et Wolfgang Weck. Component-oriented programming. In *Object-oriented technology ecoop'99 workshop reader*, pages 184–192. Springer, 1999.

[Szyperski, 2002] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd édition, 2002.

[Tatsubori et Takashi, 2006] Michiaki Tatsubori et Kenichi Takashi. Decomposition and abstraction of web applications for web service extraction and composition. In *Proc. of the IEEE ICWS*, 2006.

[Tibermacine *et al.*, 2013] Okba Tibermacine, Chouki Tibermacine, et Foudil Cherif. Wssim: a tool for the measurement of web service interface similarity. In *Proc. of CAL*, Toulouse, France, 2013.

[Tibermacine *et al.*, 2015] Okba Tibermacine, Chouki Tibermacine, et Foudil Cherif. A process to identify relevant substitutes for healing failed ws-* orchestrations. *J. Syst. Softw.*, 104(C):1–16, 2015.

[Tibermacine et Kerdoudi, 2010] Chouki Tibermacine et Mohamed Lamine Kerdoudi. From web components to web services: Opening development for third parties. In *Proceedings of the 4th European Conference on Software Architecture (ECSA'10)*, éditeurs Muhammad Ali Babar et Ian Gorton, volume 6285 de *Lecture Notes in Computer Science*, pages 480–484, Copenhagen, Denmark, 2010. Springer.

[Tibermacine et Kerdoudi, 2011] Chouki Tibermacine et Mohamed Lamine Kerdoudi. Migration d'applications à base de composants Web en services et orchestration de services Web. In *Proceedings of the french-speaking conference on Software Architectures*, page 10, Lille, France, Juin 2011.

[Tibermacine et Kerdoudi, 2012] Chouki Tibermacine et Mohamed Lamine Kerdoudi. Migrating component-based web applications to web services: Towards considering a "web interface as a service". In *Proceedings of the 19th IEEE International Conference on Web Services (ICWS'12)*, éditeurs Carole A. Goble, Peter P. Chen, et Jia Zhang, pages 146–153, Honolulu, Hawaii, USA, 2012. IEEE Computer Society.

[Tzerpos et Holt, 2000] Vassilios Tzerpos et R. C. Holt. Acdc : An algorithm for comprehension-driven clustering. In *In Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE)*, pages 258–267. IEEE, 2000.

[Upadhyaya *et al.*, 2012] Bipin Upadhyaya, Foutse Khomh, et Ying Zou. Extracting restful services from web applications. In *Proc. of IEEE SOCA*, 2012.

[Upadhyaya *et al.*, 2013] Bipin Upadhyaya, Ran Tang, et Ying Zou. An approach for mining service composition patterns from execution logs. *Journal of Software: Evolution and Process*, 25(8):841–870, 2013.

[Upadhyaya *et al.*, 2015] Bipin Upadhyaya, Ying Zou, et Foutse Khomh. An approach to extract restful services from web applications. *International Journal of Business Process Integration and Management*, 7(3):213–227, 2015.

[Yu *et al.*, 2007] X. Yu, Y. Zhang, T. Zhang, L. Wang, J. Zhao, G. Zheng, et X. Li. Towards a model driven approach to automatic bpel generation. In *Proc. of ECMFA*, 2007.

[Zeng *et al.*, 2004] Liangzhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, et Henry Chang. Qos-aware middleware for web services composition. *IEEE Trans. Softw. Eng.*, 30(5):311–327, 2004.

[Zernadji *et al.*, 2015] Tarek Zernadji, Chouki Tibermacine, Foudil Cherif, et Amina Zouioueche. Integrating quality requirements in engineering web service orchestrations. *Journal of Systems and Software*, 2015.

[Zhang *et al.*, 2006] Zhuopeng Zhang, Hongji Yang, et William C. Chu. Extracting reusable object-oriented legacy code segments with combined formal concept analysis and slicing techniques for service integration. *Proc. of QSIC 2006*, pages 385–392, 2006.

[Zhang et Yang, 2004] Zhuopeng Zhang et Hongji Yang. Incubating services in legacy systems for architectural migration. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04)*. IEEE Computer Society, 2004.