Web Service Composition Compatibility : adaptation in the presence of Business Protocol Evolution

Maryam Darvish Eslamichalandar

CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS	le cnam

École Doctorale Informatique, Télécommunications et Electronique de Paris

Centre d'Etudes et De Recherche en Informatique du CNAM

THÈSE DE DOCTORATE

présentée par : **Maryam ESLAMICHALANDAR**

soutenue le : **19 Décembre 2013**

pour obtenir le grade de : **Docteur du Conservatoire National des Arts et Métiers**

Discipline/ Spécialité : **Informatique**

Compatibilité de la composition des services Web: Adaptation suite à l'évolution des protocoles métier

THÈSE DIRIGÈ PAR:
M. BARKAOUI Kamel — *Professeur, CNAM (Cedric)*

RAPPORTEURS
M. PERRIN Olivier — *Professeur, Université de Lorraine (INRIA-LORIA)*
M. TOUMANI Farouk — *Professeur, Université Blaise Pascal Clermont-Ferrand (LIMOS)*

EXAMINATEURS
M. AKOKA Jacky — *Professeur, CNAM (Cedric)*
M. MONSUEZ Bruno — *Professeur, ENSTA ParisTech (LEI)*
Mme. SI-SAID CHERFI Samira — *Maître de conférences, CNAM (Cedric)*

To my adorable husband, Parviz,
my wonderful parents, my beloved brother, my kind uncles and my dear in laws
for their continuous supports and encouragments.

Acknowledgments

First of all, I would like to sincerely thank my supervisor, Professor Kamel Barkaoui. I genuinely appreciate for his extraordinary patience, countless supports, and invaluable guidance during my research. Without him, I never could start my doctorate research studies in Paris. His strong experience and profound knowledge in formal methods especialy in Petri nets gave me the opportunity to go through and experience this domain. Many thanks to him for everything that he has been provided me for better conducting my thesis.

My very special thanks go to Dr. Hamid Reza Motahari Nezad, who let me experience the research on service adpatation and evolution. I am grateful to him for benefiting from his incredible research expriences, remarkable supports, and invaluable ideas.

Many thanks to my dear friends Redha, Manel, Houda, Amel, Meryem, and Zeinab who have often been a welcome distraction in the research stress. Many more friends accompanied me in a wonderful and supportive way during my PhD's life in Paris form the first year and without mentioning each of them individually I would like to cordially thank them.

I would like to express my gratitude to my family especially my husband Parviz for their patience and faithfully supports.

Finally, I gratefully thank the Conservatoire National des Arts et Métiers (CNAM) and Cedric for providing the opportunity of benefiting from such a reseach environment to conduct resourcefully my doctorate studies.

Résumé

Avec l'utilisation croissante d'architectures logicielles indépendantes de la plate-forme et du langage dans le paradigme de *l'architecture orientée services* (SOA), la technologie de services web permet l'interopérabilité dynamique et flexible des processus métiers aussi bien au niveau intra qu'inter-organisationnel. Bien que la normalisation des services web permet de réduire l'hétérogénéité et rend plus facile leur interopérabilité, il y a toujours besoin de vérifier leur compatibilité en particulier dans le contexte inter-entreprises. Deux services sont compatibles si une collaboration entre eux est accomplie avec succès et que chacun puisse atteindre ses résultats attendus (états finaux). L'approche typique devant permettre à des services incompatibles d'interagir correctement est *l'adaptation* du service. L'adaptation consiste dans ce contexte à faire face principalement aux discordances relevées au niveau des *interfaces* de service (incompatibilités entre signatures de services) ainsi qu'aux discordances qui ont lieu au niveau des *protocoles métiers* (incompatibilité dans l'ordre des messages échangés entre services). On distingue deux principales techniques d'adaptation: modification de service ou synthèse d'un composant adaptateur. L'adaptation en termes de modification de service exige l'application de certaines mesures d'optimisation pour supporter les spécifications du service partenaire. Dans le cas où l'adaptation traite de la création d'un adaptateur, un composant autonome modère les interactions entre les deux services de sorte que l'interopérabilité soit obtenue. En d'autres termes, l'adaptateur compense les différences entre interfaces de services par conversion de données (c'est-à-dire par transformation de message) et celles entre protocoles métiers en réorganisant les échanges de messages ou en générant un message manquant.

Nous nous concentrons ici sur le problème de la reconfiguration dynamique de l'adaptateur en presence d'évolution de protocols métiers. Après avoir traité de la vérification d'un adaptateur en exploitant des techniques structurelles existantes développées dans le cadre de la théorie des réseaux de Petri, nous établissons une identification des patrons de mise à jour

d’adaptateurs ainsi que la mise en correspondance de ces patrons avec les différents types d’évolutions possibles au niveau des protocoles métiers des services web. Ce travail a abouti à la proposition d’un algorithme permettant, d’une part de détecter les patrons d’évolution adéquats suite à une évolution d’un des protocoles métier des services partenaires et, d’autre part et sous certaines conditions, la mise à jour à la volée de la specification du nouvel adaptateur obtenu ainsi que sa verification.

Enfin, les expérimentations réalisées sur un prototype montrent les avantages en termes de temps et de coût de l'approche dynamique proposée par rapport aux méthodes statiques conduisant systématiquement à la regeneration complète de l’adaptateur.

Mots-clés: Services Web, composition de services, compatibilité, évolution de protocole métier, adaptation.

Abstract

The advent of Web service technologies in the paradigm of *Service oriented architecture* (SOA) enables dynamic and flexible interoperation of distributed business processes within and across organization boundaries. One of the challenges in working with heterogeneous and autonomous Web services is the need to ensure their interoperability and compatibility. The typical approach for enabling incompatible services to interact is service *adaptation*. The need for adaptation in Web services comes from the heterogeneity at the levels of service *interface* and *business protocol*. The service interface incompatibilities include service *signature* mismatches (e.g., message and operation name, number; the type of input/output message parameters of operations; and the parameter value constraint). The mismatches at the business protocol (or service behavior) level arise from the order constraints that services impose on messages exchanges (e.g., *deadlock* where both partner services are mutually waiting to receive some message from the other, and *unspecified reception* in which one service sends a message while the partner is not expecting it). In service interaction through adaptation, an *adapter* mediates the interactions between two services with potentially different interfaces and business protocols such that the interoperability is achieved, i.e., adapter compensates for the differences between their interfaces by data mappings, and between their business protocols by rearranging the messages exchanges or generating a missing message.

In this dissertation, we focus on how to cope with the dynamic evolution of business protocol P of a given service (i.e., P is changed to P') that is adapted by an adapter in the context of service interaction. Web service specifications constantly evolve. For variety of reasons, service providers may change their business protocols. Therefore, it is important to understand the potential impacts of the changes arising from the evolution of service business protocol on the adapter.

We present an approach to automatically detect the effects of business protocols evolution on the adapter and, if possible, to suggest fixes to update the specification of adapter on-the-

fly. Besides, we propose a technique to verify the correctness of new adapter which is dynamically re-configured. Finally, we describe a prototype tool where experimentations show the benefits of proposed approach in terms of time and cost compared to the static methods aiming for complete regeneration of adapter or manual inspection and adaption of the adapter with respect to changes in the business protocols.

Keywords: Web services, compatibility, service composition, business protocol evolution, adaptation.

Table of contents

List of tables

List of figures

Introduction

1 Context

With growing the idea of abstracting from underlying technologies and implementations, *Service-Oriented Computing* (SOC) [1] has been emerged as a paradigm of cooperation of self-described software components called *services*. Services are open components and self-describing encapsulation of business functionality that support rapid, low-cost development and deployment of distributed applications. A service has an identifier and can deliver its functionality via a standardized interface [2]. The most prominent kind of services is *Web services* [3]. A Web service[1] defined by World-Wide Web Consortium (W3C) is *a software application identified by a URI, whose interfaces and binding are capable of being defined, described, and discovered by XML artifacts, and supports direct interactions with other software applications using XML-based messages via internet-based protocols* [4]. Web services can potentially simplify the building of distributed applications [5]. Standards like XLANG[2] or Web Services Business Process Execution Language (WS-BPEL) [6] allow realizing the business processes using Web services. Web services have been widely adopted to practically implement the fundamental idea of *Service-Oriented Architecture* [7-8]. Service-Oriented Architecture (SOA) is an approach to the development of loosely-coupled, protocol independent and distributed software applications as a collection of well-defined autonomous services within and across enterprises in a standardized way. Such standardization enhances re-usability and reduces considerably the cost of application integration. The idea of SOA is to use services as loosely-coupled interfaces and thereby hiding implementation details [9]. The primary advantages of the SOA comprises of *reusability*, *interoperability*, *scalability*, *flexibility*, and *cost efficiency*. The SOA is often realized through Web Services technologies [10].

Web services technologies as illustrated in the Figure 1.1 include:

– XML Technologies comprising base XML (Extensible Markup Language)[3], XML Schema[4], XSLT, XPATH, XQUERY to support the data and document that is exchanged, e.g., transformation, process, manipulation, and etc.

[1] In this thesis, we use the terms “Web service” and “service” interchangeably
[2] Thatte S., XLANG: Web Services for Business Process Design, Technical report, Microsoft, 2001
[3] http://www.w3.org/XML.
[4] XML Schema provides the way to define an XML document type and structure

- Simple Object Access Protocol (SOAP) [11] that is a standard enabling the basic exchange of data and documents (i.e., in XML format) between two or more peers. SOAP is used for transportation with different Internet protocols such as HTTP, SMTP, FTP, and so on.
- Web Services Description Language (WSDL) [12] as an XML-based language describes all details about Web service functionalities, and interfaces. In addition, WSDL describes where the service is located, and how to access the service.
- Universal Description, Discovery and Integration (UDDI) [13] define the standard infrastructure for publishing, discovering and integrating services. UDDI is just a registry enabling service requesters to look for Web services based on their functional specification but not quality information.

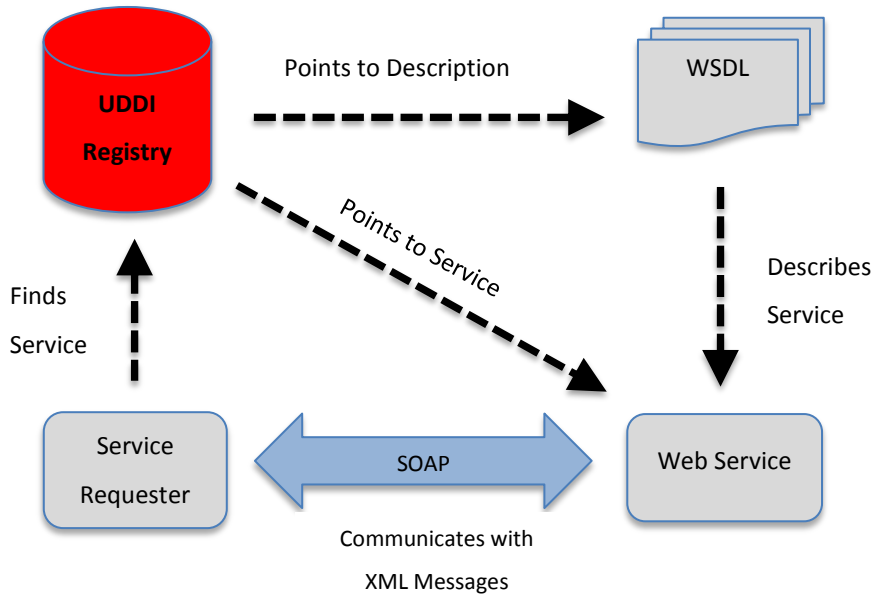

Figure 1. Web services technologies

As Figure 1.2 illustrates, the SOA provides a general framework for Web services collaboration comprising 1) *service provider* that offers his own Web service (i.e., by generating WSDL) and publishes the required information to a public/private *service repository*; 2) *service broker* that is responsible for managing the repository and allows the service requester to find an appropriate service (that has already been published) for binding to his own components; and 3) *service requester* that invokes the services. The service broker compares the actual Web service model published by provider with an abstract Web service model submitted by the service requester, such that they can directly bind their services and initiate the interaction. In some cases the broker may construct an adapter service to bridge the gap between the provider and requester. There are two ways of binding of Web services: *static* binding that is done at design time where the service requester acquires required information about a service directly from the service

provider (i.e., through a private channel like e-mail), and stores it in a local configuration file; and *dynamic* binding that occurs at runtime.While the client application is running, it dynamically locates the service using a UDDI registry and then dynamically binds to it using WSDL and SOAP. This requires that the contents of the UDDI registry be trusted. Currently, only private UDDI networks can provide such control over the contents. Felber et al. in [14] introduce some algorithms to provide solutions for the problems associated with SOAP and related XML-based schemes that require further decoding to bind many kinds of requests.

The mature standards like WSDL and ebXML Collaboration Protocol Profile [15] are used for Web service descriptions. Web Service description generally represents service *capability*, *behavior* (*business protocol*), *quality*, and *interface*. An instance of a given service corresponds to the execution of its activities. These activities are atomic units of work where the partial order of execution of the activities denotes the behavior of a service. The expected results and conceptual purpose of a service signifies its capability. The quality of a service (QoS) is realized by non-functional properties. A service interface represents its functionality. The interface describes service *signature* (i.e., comprising the operations, the inputs/outputs messages, message types, and error parameters) facing with an environment from interactional standpoint. The interface of a service consists of a set of ports (i.e., connected by a channel) enabling the message exchange for a service using SOAP over the transport protocol such as HTTP or HTTPS. Web services interfaces are usually described using WSDL.

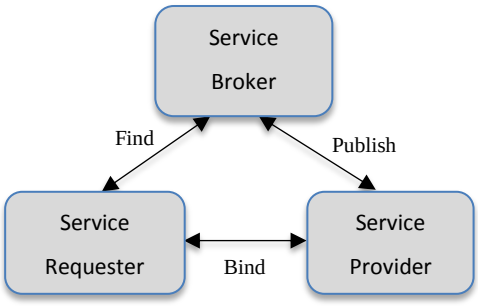

Figure 2. The SOA framework

In the past decade, numerous approaches have been devoted to different aspects of Web services associated with modeling, interoperability, quality of services (QoS), semantics, security, and automation of different processes such as discovery, selection, and composition. Among these challenges, our main concern in this dissertation[5] is service composition[6]. Web service composition as the fundamental idea of SOA is to combine individual and simple services into complex processes to provide dynamic, automatic, seamless service interoperability. Composition of Web services has grown to support B2B or enterprise

[5] We use the terms “dissertation” and “thesis” interchangeably
[6] We use the terms “composition”, “interaction”, “interoperability”, “collaboration”, and “integration” interchangeably

application integration. Service composition establishes a value-added functionality by integration of various services conducted by different organizations such that may not be foreseen at the time when a Web service is designed. By Web service composition, reuse of services is implemented, the quality and efficiency are increased, and the dynamic evolvement of the user's requirements is satisfied. Lots of work on Web service composition has been presented by focusing on different aspects of Web service specifications in the literature [16-17]. The process of service composition develops a *composite* service by taking a set of service components as input. A composite service is recursively defined as an aggregation of elementary and composed services [16].

Motahari-Nezhad et al. [18] discuss the specifications of Web services interoperability based on layers concepts for better investigation of the relevant problems in different layers. The specifications in lower layer are required properties in most applications while the higher-layer ones are not necessary:

- *Messaging layer*. In addition to the SOAP as a common protocol, in this level service providers may consider the desirable properties such as security or reliability as additional service specifications.
- *Basic coordination* (WS-Transaction). These properties define the specifications associated with messages exchanges among partner services (e.g., federate security management that is useful in many business processes developments).
- *Business-level interfaces and protocols*. This layer includes the functional properties of services (i.e., the service interface and service business protocol).
- *Policies and non-functional aspects*: The specifications of this layer comprise of security policy, the QoS characteristics such as time-related properties and cost.

In this thesis, we deal with the business-level interfaces and protocols of Web service at higher abstractions levels of the service interaction stack.

Due to the heterogeneity and autonomy of Web services, the necessity of evaluation and verification the composite services remains as a fundamental challenge for ensuring the correct composite services. Even though service providers prevent the publication of erroneous services, nevertheless for service customer it is crucial to verify the correctness of a composite service before running (like *model-checking*). Verification of service composition is inevitable to avoid the great economic loss while checking whether the designer's requirement or users' needs are satisfied. Verification approaches analyze the behavior of a composite service based on a variety of correctness criteria and different formalisms.

A fundamental challenge in verification of collaborative services is concerned with the *compatibility* among participant services [19]. It is crucial that the interoperability of heterogeneous Web services to be compatible. Two services are said to be compatible if any collaboration between them is accomplished successfully, and also each of them reaches its expected results (final states). A variety of research efforts on Web services have fundamentally focused on dealing with incompatibilities of service interaction. The typical approach for enabling incompatible services to interact is service *adaptation* [20-22]. The need

for adaptation in Web services comes from the heterogeneity at the levels of service *interface* and *business protocol*. The service interface incompatibilities include service *signature* mismatches (e.g., message and operation name, number; the type of input/output message parameters of operations; and the parameter value constraint). The mismatches at the business protocol (or service behavior) level arise from the order constraints that services impose on messages exchanges, e.g., *deadlock* where both partner services are mutually waiting to receive some message from the other, and *unspecified reception* in which one service sends a message while the partner is not expecting it.

In service interaction through adaptation [20] [23] an adapter mediates the interactions between two services with potentially different interfaces and business protocols such that the interoperability is achieved, i.e., adapter compensates for the differences between their interfaces by data mappings, and between their business protocols by rearranging the messages exchanges or generating a missing message. In this dissertation, we focus the adaptation of services from viewpoint of their business protocols and abstract from any other aspects such as non-functional properties (e.g., time constraint, cost), data and information semantics.

2 Contributions

Existing service adaptation approaches support the business protocols that do not change after adapter generation. For variety of reasons, service providers may change their business protocols. One interesting challenge in service adaptation is to cope with the dynamic evolution of business protocol P of a given service (i.e., P is changed to P') that is adapted by an adapter. Web service evolution is usually driven by *perfective* motivation – to modify the existing functionality of services or business rules; and also by *corrective* motivation – to change the service signature, behavior or policy [24]. Implementing these changes causes challenges in many aspects of managing changes in adapters. An interesting issue is to understand the impact of the changes on the specification of the existing adapter. A baseline approach would be to re-generate adapter from scratch whenever there is a change on business protocols of the partner services. Such an approach is not efficient in the case where changes of business protocols have no global impact. Instead, a more beneficial approach is to analyze the potential impacts of these changes on the adapter supporting the interactions among services. Therefore, there is a need a method to detect the evolution of business protocols participating in an adapter, and identify whether an adapter re-generation is needed, or alternatively the changes can be remediated on-the-fly.

In this thesis, we focus on the problem of service adaptation when the business protocols of services evolve and move to a new version. The challenges include identification of the changes on the business protocols, and their respective impacts on the adapters. Such an analysis would need to distinguish changes that do not have any impact on the adapter, or can be accommodated in a dynamic change to the adapter specification via approaches such as software aspect [25], or they would require a re-generation of the

adapter. However, updating the adapter specification at run-time may be preferable than a complete regeneration. The intention is to provide enough information to adapter developers in deciding how to deal with changes in the underlying interacting services.

In this dissertation, we aim to investigate dynamic adaptation of an adapter in the context of business protocol evolution. In order to describe, model and analyze the evolution of business protocols we use the *open Workflow Net* (oWFN) [26] formalism that is a sort of Petri nets [27] well suited to cope with control flow dimension of the evolution of service business protocol.

We first present a method to identify the changes in service protocols participating in an adapter. The proposed method enables us to detect which elements of a given business protocol have been modified, i.e., messages or activities *added*, *removed* or *updated*. Then, we identify the potential impacts of the changes on the current adapter in terms of either *partial* impact (i.e., dynamically treatable changes) or *global* impact (i.e., the changes which need adapter re-generation to be treated). For the changes with partial impact, we find and suggest updating the specification of current adapter dynamically. We also verify the correctness of the new adapter which is dynamically re-configured.

In particular, we make the following contributions that summarize the major points of our work:

Introducing Adapter Adaptation Patterns

We identify and classify the possible common *adapter adaptation patterns* (AAPs) for business protocols evolution. For this end, we address the taxonomy of these AAPs with respect to the type of changes that may occur at the level of service's interface elements (e.g., messages or activities *added*, *removed* or *updated* in different points of the interactions).

Therefore, each of the AAPs characterizes a business protocol *evolution pattern* (i.e., the change type) and describes the *potential impact* of an evolution pattern on the adapter which can be partial or global impact. An AAP also recommends a *strategy* for updating the adapter specification on-the-fly, if the impact of respective evolution pattern is partial and such a solution exists.

Protocol Evolution Patterns Identification

We present an algorithm to detect the changes of business protocols in terms of the evolution patterns (of the AAPs). The algorithm allows recognizing the elements of business protocol that have changed in terms of added, removed, and updated interface elements.

Protocol Evolution Impact Analysis on the Adapter

We introduce an algorithm to automatically analyze the impact of evolution in business protocols by checking the AAPs. The impact analysis algorithm explores the affected areas of business protocol using a bfs-based method (*breadth-first search*) and evaluates the potential influences of changes by checking the corresponding AAPs. For each of the involved AAP with partial impact, the algorithm dynamically applies the adaptation strategies on current adapter to re-configure the adapter specification. Otherwise, the algorithm recognizes that the adapter should be completely re-generated from scratch – using a global analysis of the new interactions among services for the evolved business protocols.

Adapter Verification

We use a technique based on the structure theory of Petri Nets [28] to dynamically verify the correctness of the adapter which is updated on-the-fly. Indeed, it is crucial to verify the *well-formedness* [20] property of generated adapter. An adapter is said to be well-formed if it supports a compatible collaboration between partner services. We show how we can take the advantages of concepts based on the structure theory of Petri nets to check whether the new adapter satisfy the well-formedness property.

Prototype Implementation and Experiments

Finally, we present a prototype tool implementing the proposed approach based on the PIPE2 (Platform Independent Petri Net Editor 2) [29]. The obtained experimental results show that adapter developers can save a significant amount of time, cost and efforts by applying this approach.

To the best of our knowledge of the state of the art, this work is the first to deal with the dynamic re-configuration of the adapter in the context of evolution of partner service's business protocols. The proposed approach is useful for service clients with adapters in place when they receive notices of changes in the partner business protocols in order to analyze the impact and ensure compatibility, and identify possible updates on the adapter specification to remedy the changes.

As we already pointed out, in this work we leave out the discussion about Web service non-functional requirements (time constraint, cost), data and information semantics. Hence, a Web service can be viewed as a control structure describing its behavior according to an interface to communicate asynchronously with other services. Parts of this thesis have been published in earlier papers [30-31] [150].

3 Outline

This dissertation proceeds as follows. In chapter 1, we present the basic concepts and some background on Web service definition and modeling. Chapter 2 is devoted to the Web service composition. It discusses the verification of service composition and the notion of compatibility. In chapter 3, we present the need of adaptation for service interaction and review a number of proposed adaptation approaches mainly from points of view of their interfaces or business protocols mismatches.

Chapter 4 is devoted to the issue of Web service evolution and related concepts. This chapter mainly presents our proposed approach for adaptation of Web services in the case where the business protocols of services evolve. Therefore, in this chapter we define the common adapter adaptation patterns in presence of business protocol evolution. Besides, we present a method for identifying these patterns. We also introduce an algorithm to analyze the changes in business protocols participating in an adapter and to apply the adaptation solutions on it. In addition, chapter 4 illustrates a technique to verify the correctness of the new adapter. We also discuss the implementation of a prototype and experiments in this chapter. In chapter 5, we conclude and describe some future perspectives.

Chapter 1

Web Service Definition and Modeling

Emerging technologies and industrial standards in the context of Web services facilitate dynamic and flexible cooperation of distributed business processes within and across organization boundaries to develop value-added applications. Web service is a functionality described in a standard definition language and can be accessed via various transport protocols. In this chapter, we provide an overview of the necessary formal background for modeling the Web service specifications that we apply in this dissertation.

1.1 Background and Preliminaries

Service-orientation specifies common descriptions and definitions for Web services such that each service has an internal control and an interface [32] to expose a given functionality to its environment. An interface of a Web service usually consists of the *message channels* that are exposed to the environment (i.e., to invoke other services). The message channels group into ports. Using each port, the interface is able to store messages. In a port, each message channel has a direction and is either an *input* message channel or an *output* message channel. The interfaces can be composed by connecting these message channels that make the asynchronous communication in which sending and receiving of messages is decoupled. In BPEL the term *partner link* has been attributed for a port of an interface. The internal control triggers the actions of the Web service. An action either operates locally or responsible to send a message to a port or receive one from a port.

Service business protocol

The *business protocol* of a service (service *behavior*) describes the external behavior of service exposing to the environment. In other words, service business protocol[7] is not concerned with the execution of internal operations. Indeed, the behavior of a service specifies the order in which messages or documents are exchanged (i.e., message exchange sequences). The Figure 1.3 depicts an example of a service protocol in the forms of open Workflow Nets.

The service behavior can be expressed using standard languages such as BPEL (Business Process Execution Language) [6] or a modeling diagram such as *state diagrams* [32].

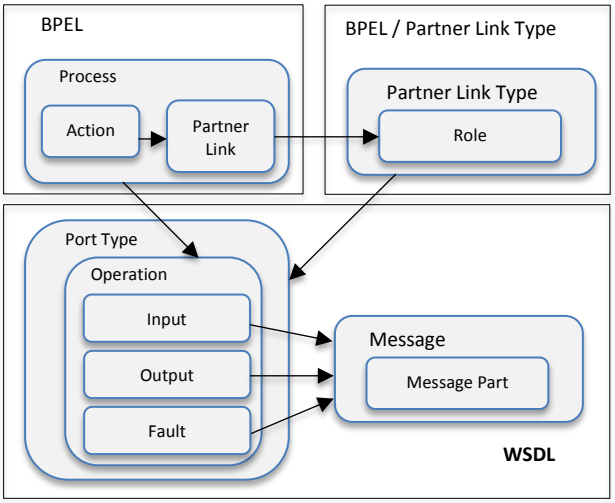

Figure 1.1. The relation between BPEL process definition and the WSDL

Industrial standards

Industrial efforts are principally dedicated to development of XML-based languages on the top of UDDI, WSDL and SOAP for description of complex business processes and Web services interactions.

The Business Process Execution Language for Web services (BPEL4WS or BPEL for short) [6] is an XML-based standard language to describe the behavior of Web services in a business process interaction. Originally, the BPEL has been released by Microsoft, IBM, Siebel Systems, BEA, and SAP by merging the specification of Microsoft's XLANG and IBM's WSFL[8] in July 2002. BPEL is now supported by the OASIS group. Due to its expressiveness, BPEL is currently seen as a fundamental standard. BPEL is used to model two kinds of *executable processes* and *abstract processes*. An abstract process describes the interaction protocol and message exchange between the involved partners without revealing the internal

[7] In this dissertation, we use the term "business protocol" and "protocol" interchangeably.
[8] Leymann F., Web Services Flow Language, Technical report, IBM, 2001.

behavior of them. The executable process essentially models a workflow where specifies the execution order of number of activities. An activity can either be a *primitive* activity (e.g., *invoke*, *reply*, *receive*, *wait*, *assign*) or a *structured* activity. The structured activities are a combination of primary activities in order to enable the presentation of complex structures such as *sequence*, *switch*, *while*, *flow*. As shown in Figure 1.1, the BPEL process model places on top of WSDL.

Besides, Business Process Modeling Language (BPML)[9] proposed by Business Process Management Initiative. In parallel, the W3C has provided a common framework to *Semantic Web* idea [33] that allows resources to be shared and reused across application, enterprise, and community boundaries.

Industrial Web service standard languages often insufficient for modeling complex process and require theoretical support. Therefore, various formal modeling are used for representing the BPEL activities. Petri nets [34-38]; Finite State Machines (FSMs) [39-40]; process algebra [41-42]; π-calculus [43] are effectively applied to model, analyze, and verify the behavior of Web services and service interoperability. Among these formalisms, Petri nets (P/T nets) as well-appropriate techniques representing the behavior of concurrent and non-deterministic systems are effectively applied for modeling Web services specifications.

1.2 Formal Model

1.2.1 Petri nets

In this section, we provide an overview on the basic definitions and properties of Petri nets while giving some structure theory of P/T nets.

1.2.1.1 Definitions and specifications

Originally, Petri net has been proposed by Carl Adam Petri [44]. The classical Petri net is a directed, connected, and bipartite graph with two node types called *places* and *transitions*. These nodes are connected by directed *arcs*. Due to the intuitive graphical and visualized representation of Petri nets, they are broadly exploited in both theoretical and practical research studies. The Petri nets can be automatically extracted from the BPEL [45]. Variety of Petri net tools[10] exist that each of them has unique internal representation and file format definition, e.g. Artifex, CPN-AMI, HiQPN-Tool, Renew, PACE. Moreover, Petri Net Markup Language (PNML)[11] has been proposed as an XML-based interchange format for supporting different versions of Petri nets.

[9] Arkin, A. 2002. *Business Process Modeling Language (BPML)*, http://www.bpmi.org/
[10] http://www.daimi.aau.dk/PetriNets/tools
[11] http://www.informatic.hu-berlin.ed/top/pnml

Definition 1.1 (Petri Net). A Petri net N is a triple (P, T, F) in which

- Two finite and disjoint sets P of places and T of transitions such that $P \cap T = \varnothing$ and $P \cup T \neq \varnothing$;
- F is set of arcs with $F \subseteq (P \times T) \cup (T \times P)$ as the flow relation. Indeed, F is a function that allocates to each transition its pre-set places and post-set places.

A transition $t \in T$ is an active element representing an activity of a service, and a place $p \in P$ is a passive element standing for a message or data. Transitions are represented by rectangles and places by circles. At any time a place contains zero or more *tokens*. Graphically, a token is depicted by a black dot. Tokens in the Petri nets can represent the resources and resource allocation. A *marking m* (i.e., the state) of a Petri net is the distribution of tokens over places (i.e., m: $P \rightarrow \mathbb{N}$) that can be represented as $2p_1 + p_2 + 1p_3 + \ldots + kp_n$ ($k \geq 0$ and n is the number of set P). Hence, m_0 represents the initial marking and m_f stands for the final marking. Besides, $m(p)$ demonstrates the number of tokens in the place p.

The pre-set and post-set of a node $x \in P \cup T$ describe the set ${}^\bullet x = \{y \in P \cup T \mid (y, x) \in F\}$, and the set $x^\bullet = \{y \in P \cup T \mid (x, y) \in F\}$, respectively. A transition $t \in T$ is said to be *enabled* in marking m if and only if $\forall p \in {}^\bullet t$ holds $m(p) > 0$ (i.e., each pre-set place of t contains at least one token). An enabled transition t can be *fired* in marking m by consuming tokens from each pre-set place $p \in {}^\bullet t$ and producing tokens on the all post-set places $p \in t^\bullet$ that results in a new marking m' denoted by $m \xrightarrow{t} m'$ such that $m' = m - {}^\bullet t + t^\bullet$ (i.e., $m'(p) = m(p) - 1, \forall p \in {}^\bullet t$; and $m'(p) = m(p) + 1, \forall p \in t^\bullet$).

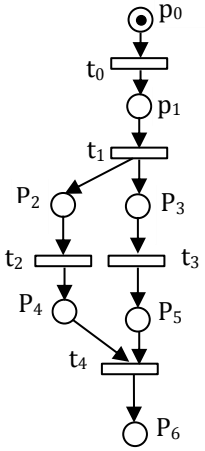

Figure 1.2. Example of a Petri Net

A sequence σ over T is a function σ: $\{1, ..., n\} \rightarrow T$, by length $|\sigma| = n$ ($n \in \mathbb{N}$). A marking m_n is called *reachable* form marking m_1 if and only if there is a firing sequence $\sigma = t_1 t_2 \ldots t_{n-1}$ such that $m_1 \xrightarrow{\sigma} m_n$ (i.e. $m_1 \xrightarrow{t1} m_2 \xrightarrow{t2} m_3 \ \ldots \xrightarrow{tn} m_n$). A pair (N, m_0) represents a marked Petri net N. The set of all reachable markings of a Petri net (N, m_0) is defined as $R(N, m_0)$. We also denote by $\|m\| = \{p \in P \mid m(p) > 0\}$ the support of marking m. We consider $\mathbb{N}^P$ for the set of all markings over P. Figure 1.2 shows a simple

example of a Petri net. In this thesis, we use ordinary Petri nets, which do not allow multiple arcs between places and transitions.

Definition 1.2 (incidence matrix). The incidence matrix of the Petri net N is the matrix C indexed by $P \times T$ and defined by $C(p, t) = W(t, p) - W(p, t)$ in which $W(u) = 1$ if $u \in F$ and $W(u) = 0$ if $u \notin F$.

The notion of place invariants and transitions invariant refers to the conservation of tokens and sequences without effect respectively. A non-negative integer vector f ($f \neq 0$) indexed by P ($f \in Z^{|P|}$) is a *P-invariant* if and only if it satisfies ${}^{t}f.C = 0$ (0 is a column vector where every component equals 0). A non-negative integer vector g ($g \neq 0$) indexed by T ($g \in \mathbb{N}^{|T|}$) is a *T-invariant* if and only if it satisfies $C.g = 0$. P-invariant characterizes that the number of tokens over a set of places P of a Petri net N is constant independently from any transition firing. We denote by $\| f \| = \{p \in P \mid f(p) \neq 0\}$ the support of f such that $\| f \|^{+} = \{p \in P \mid f(p) > 0\}$ and $\| f \|^{-} = \{p \in P \mid f(p) < 0\}$ are the positive and negative support of f respectively.

We recall some behavioral properties of a Petri net N as follows:

- A marking m^{*} is a *home state* if and only if $\forall m' \in R(N, m)$, $m^{*} \in R(N, m')$.
- N is *reversible* if and only if m_0 is a home state.
- N is *bounded* if and only if for every place $p \in P$ and for every reachable marking $m \in R(N, m_0)$, $\exists k \in \mathbb{N}$ such that $m(p) \leq k$.
- N is *structurally bounded* if N is bounded for any m_0.
- N is *conservative* if and only if there is a P-invariant f such that $\| f \| = \| f \|^{+} = P$. In other words, a Petri net is conservative if the total number of tokens presented in the set places P of the net is constant in any reachable marking. If N is conservative then N is structurally bounded.
- N is *live* if and only if for every transition $t \in T$ and for every reachable marking $m \in R(N, m_0)$, there is a reachable marking $m' \in R(N, m)$ in which t is enabled.
- N is *quasi-live* if and only if $\forall t \in T$, $\exists m \in R(N, m_0)$ in which t is enabled.
- A transition t of N is *dead* if and only if for every reachable marking $m \in R(N, m_0)$, t cannot be enabled.
- N is *deadlock-free* if and only if for every reachable marking $m \in R(N, m_0)$, $\exists t \in T$ enabled in m and the final marking is reachable.
- N is structurally live if and only if $\exists m_0$ such that (N, m_0) is live.
- A bounded and live Petri net is said to be well-formed.
- N is *weakly terminating* if and only if for every reachable marking $m \in R(N, m_0)$, the final marking m_f is reachable, i.e., $m_f \in R(N, m)$. Weak termination does not require that every transition to be quasi-live.

1.2.1.2 Basics of structure theory of P/T nets

Structure theory of Petri nets investigates the relationship between the behavior and structure of the net. The use of structural methods for the analysis of systems compared to other approaches avoids the state explosion problem inherent in concurrent systems.

Definition 1.3 (Reachability graph). The *reachability graph* of a Petri net is the part of the transition system reachable from the initial place in graph-like notation.

Definition 1.4 (Siphon). A remarkable substructure of Petri nets is *siphon*. A nonempty place set $S \subseteq P$ of a P/T net N is called a siphon if and only if ${}^{\bullet}S \subseteq S^{\bullet}$. It means every transition that has inputs from one of the places in S, as well has outputs to one of the places in S. S is said to be *minimal* if and only if it contains no other siphon as a proper subset. Due to its structure, the number of tokens in a siphon will never increase, and an unmarked siphon will always remain empty. In such a case, the transitions of $S^{\bullet}$ are dead, and so S needs to be controlled. S is said to be *controlled* if and only if S is marked at any reachable marking (i.e., $\forall m \in R\,(N, m_0)$, $\exists p \in S$ such that $m(p) > 0$).

Definition 1.5 (CS-property). N is said to be satisfying the *controlled-siphon property* (CS-property) if and only if all its minimal siphons are controlled.

We recall below two well-known basic relations between liveness and the CS-property [46]. The first states that the CS-property is a sufficient deadlock-freeness condition while the second states that the CS-property is a necessary liveness condition.

Proposition 1.1. *Let N be a P/T net. If N satisfies the CS-property then N is deadlock-free* (*weakly live*).

Proposition 1.2. *Let N be a P/T net. If N is live then N satisfies the CS-property.*

The following proposition recalls two structural (sufficient but not necessary) conditions permitting us to check if a given siphon is controlled or not.

Proposition 1.3. *Let S be a siphon of N satisfying one of the two following conditions, then S is controlled* [46]:

i) $\exists R \subseteq S$ such that $R^{\bullet} \subseteq {}^{\bullet}R$ and $R \cap \|m0\| \neq \varnothing$ (i.e. $\exists p \in R$, $m_0(p) > 0$);

ii) $\exists P$-*invariant* $f \in Z^{P}$ such that $S \subseteq \| f \|$, $\| f \|^{+} \subseteq S$ and , and $\sum_{p \in P} [f(p).\, m_0(p)\,] > 0$.

Definition 1.6 (K-Systems). P/T nets where CS-property is not only necessary but also sufficient liveness condition are called *K-systems*. In other words, there is equivalence between liveness and CS-property in K-systems [46].

Definition 1.7 (Root place). Let $t \in T$ be a transition of a Petri net N, $r \in P$, and $r \in {}^\bullet t$; r is a *root place* for t if and only if $\forall p \in {}^\bullet t, r^\bullet \subseteq p^\bullet$.

Definition 1.8 (Ordered transition). A transition $t \in T$ is said to be ordered if and only if $\forall p, q \in {}^\bullet t$, $p^\bullet \subseteq q^\bullet$ or $q^\bullet \subseteq p^\bullet$. An ordered transition has at least one root place. A transition admitting a root place is not necessarily ordered.

Let us to consider the following denotes:

- *Root* (t): the set of root places of t.
- T_O (N): the set of ordered transitions of N.
- T_R (N): the set of transitions of N admitting a root (i.e. T_O (N) $\subseteq T_R$ (N)).
- *Root* (N): the set of root places of N.

Definition 1.9 (Root Component). The *Root Component* of N is the net $R_C(N) = (P_C(N), T_C(N), F_C(N))$ where P_C = *Root* (N); $T_C = T_R$ (N); and F_C is the restriction of F such that $(p, t) \in F_C$ if and only if $p \in Root(t)$, and $(t, p) \in F_C$ if and only if $(t, p) \in F$.

Two main subclasses of K-Systems namely *Root nets* and *Ordered nets* can be recognized structurally and effectively [46]. Note that by definition these two subclasses are disjoint.

Definition 1.10 (Root net). N is called a Root net if and only if $T_R(N) = T$ and its Root Component $R_C(N)$ is bounded and connected.

Definition 1.11 (Ordered net). N is called an Ordered net if and only if T_O (N) = T (i.e. all its transitions are ordered).

Theorem 1.1. *Let N be an Ordered net or a Root net. N is live if and only if it satisfies the CS-property* [46].

In the following, we introduce the *open nets* (*open WorkFlow Net*) [26][47][48] formalism that we apply in this thesis as a basis for modeling and analyzing Web services specifications and services

interactions. This formalism is especially exploited for representation and evaluation of Web service business protocol evolution as the main challenge of this dissertation.

1.2.2 open WorkFlow Nets

In this section, we describe the *open Workflow Net* (oWFN) formalism to efficiently present our approaches. An oWFN is a type of Petri nets along with interface places for asynchronous communication with the partners. Indeed, an oWFN encompasses a *workflow net* describing the internal process of a Web service, and *input* / *output* interface places enabling the exchange of messages sent or received by a service.

We recall that a workflow net is a Petri net with two distinguished places – the initial place p_i (i.e., with empty pre-set), and the final place p_f (i.e., with empty post-set) – and that for every node $n \in P \cup T$, there is a directed path from place p_i to p_f via n [49].

In many references the interface places are considered as data places. More precisely, each input place (i.e., with empty pre-set) corresponds to an input port of the interface (i.e., used for receiving messages from a distinguished channel) whereas an output place (i.e., empty post-set) corresponds to an output port of the interface (i.e., used for sending messages via a distinguished channel). The Web services can be composed by connecting these interface places.

Definition 1.12 (open WorkFlow Net). An open Workflow Net (oWFN) $N = (P, In, Out, T, F, m_0, M_f)$ consists of :

- Finite and disjoint sets P of internal places, In of input interface places, Out of output interface places, and T of transitions;
- A flow relation $F \subseteq (P \times T) \cup (T \times P) \cup (T \times Out) \cup (In \times T)$;
- An initial nonempty marking m_0 such that $\forall p \in In \cup Out, m_0(p) = 0$; and
- A nonempty set of final markings M_f such that:
  - $\forall m_f \in M_f$ and $\forall p \in In \cup Out, m_f(p) = 0$;
  - $\forall p \in \|m_f\|, p^\bullet = \varnothing$ where $\|m\| = \{p \in P \mid m(p) > 0\}$ is the support of marking m; and
  - The final markings are mutually exclusive.

A transition $t \in T$ is an *interface transition* if and only if the pre-set or post-set of t includes an interface place (i.e., $|t^\bullet \cap Out| \neq \varnothing$ or $|{}^\bullet t \cap In| \neq \varnothing$). For an oWFN, we denote by T_I the set of all interface transitions, and by P_I the set of all interface places $In \cup Out$. For every interface place $p \in In$, $q \in Out$ holds ${}^\bullet p = \varnothing$ and $q^\bullet = \varnothing$.

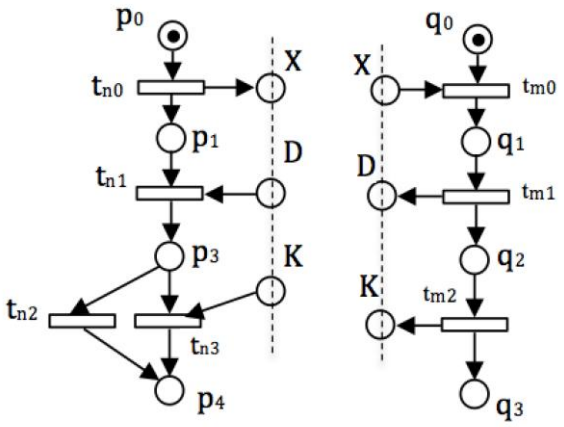

Figure 1.3. Example of two oWFNs *N1* (left net) and *N2* (right net)

Here, we consider the oWFNs where for each interface transition t holds ($|{}^\bullet t \cap In| = 1$ and $|t^\bullet \cap Out| = \varnothing$) or ($|t^\bullet \cap Out| = 1$ and $|{}^\bullet t \cap In| = \varnothing$). In addition, our approach assumes that oWFNs are acyclic (i.e., which do not contain directed cycles), hence, we can assign a topological ordering (#) to interface transitions (i.e., if t_b can be reached by t_a then $\#t_a < \#t_b$). Besides, we assume that business protocol evolution in terms of added or removed interface transitions do not create cycle in the internal part of associated oWFN. Figure 1.3 represents a simplified example of an oWFN. As showed in the Figure a dash line distinguishes the interface places. The tool BPEL2oWFN [50] automatically transforms the BPEL to an oWFN.

In the context of oWFNs cooperation, we have to respect the non-visibility of the internal part of oWFNs. Accordingly, we define the *incidence matrix* [27] of an oWFN only by the interface elements. The columns and rows of the matrix include only the input/output interface places and transitions of an oWFN, respectively.

Definition 1.13 (Incident Matrix of oWFN). For an oWFN N, the incident matrix IM is a $|P_I| \times |T_I|$ matrix where for each $p \in P_I$, $t \in T_I$, $i \in In$, and $o \in Out$;

- $IM[p, t] = 1$ if $(i, t) \notin F$ and $(t, o) \in F$;
- $IM[p, t] = -1$ if $(i, t) \in F$ and $(t, o) \notin F$; and
- $IM[p, t] = 0$ if $(i, t) \notin F$ and $(t, o) \notin F$.

Algorithm 1 represents the method of generation of the incidence matrix of an oWFM.

Algorithm 1: Generating the interface incidence matrix of oWFN *S*

Input: oWFN *S*

Output: IM^S

```
1  begin
2      F_I ← {(x, y) | (x, y) ∈ ((T_I × P_I) ∪ (P_I × T_I))};
3      m = |P_I|;
4      n = |T_I|;
5      IM[m][n] ← 0;
6      for each (x, y) ∈ F_I do
7          if (x = transition) ∧ (∃p ∈ P_I [i] | p = y) ∧ (∃t ∈ T_I [j] | t = x) then
8                  IM[i][j] ← +1;
9          else IM[i][j] ← –1;
10 return IM[m][n];
```

Chapter 2

Web Service Composition and Compatibility Analysis

Service-orientated architecture (SOA) has been emerged as a computing paradigm to generate complex services and large distributed systems by composition of several heterogeneous, decentralized, and autonomous Web services. In the past decade, the interoperability of Web services [51] as the fundamental idea for effective B2B integration [52] and satisfying business requirements has received significant attention from both industry and academia, e.g., *extended enterprise* (*business-to-business*) *pattern* introduced by IBM[12].

Web service interoperability aims to provide seamless and automatic communication among independent services regardless of their location and platform. The ability of composition of heterogeneous and autonomous Web services increases the reusability and quality of services, reduces costs, and can satisfy a variety of business demands. The process of service composition develops a composite service by taking a set of service components as input. A composite service is recursively defined as an aggregation of elementary and composed services [16]. The composition of two or more services generates a new service providing both the original behavior of initial services and a new collaborative behavior to carry out a new composite task. This means that existing services are able to cooperate even though the cooperation was not designed in advance [53] [18].

There are two different service descriptions for a composition model: service *orchestration* and service *choreography* (also called *contract*) [54]. Although both are applied to model composite services in the SOA, service orchestration (see Figure 2.1(a)) describes the service interaction from the viewpoint of a single participant service (orchestrator), while service choreography (see Figure 2.1(b)) describes the interactions between a collection of participant services from a global perspective such that each party

[12] IBM-WebSphere: Introduce Patterns for e-business.

knows the business logic and messaging sequence. By choreography, we refer to the messages exchanges that occur among partners in a service interaction. The languages like BPEL and BPMN[13] describe service orchestrations. Various specification languages also exist for services choreography such as WS-CDL [55] and BPEL4Chor [56]. Yushi et al. [57] present a survey on Web service composition languages.

Wil van der Aalst et al. [58] have proposed a public-to-private approach, in which the partners involved in inter-organizational cooperation describe a common public choreography comprising a set of the task that each task is associated with an organization. This common schema is defined as a *contract* based on predefined rules among the partners. Based on this public contract, organizations can develop and implement their internal plan (private view) autonomously and preserve their privacy by considering contract agreements. In some approaches [59-61], the authors apply a logic based framework using Linear Temporal Logic (LTL) [62] to specify, verify and validate the service composition logic.

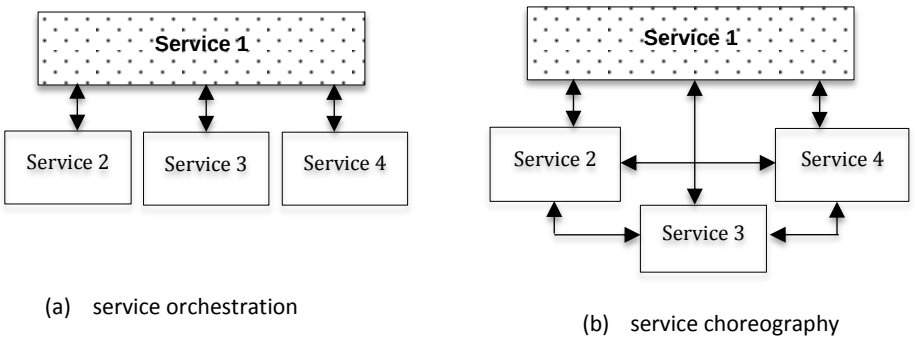

Figure 2.1. Service orchestration vs. service choreography

2.1 Research Efforts on Service Composition

In this section, we intend to provide a quick overview on the proposed Web services composition approaches.

Papazouglou M.P. [2] introduces an *extended* SOA, which includes separate tiers for service *composition*, *coordination*, and service *operations management* (i.e., to monitor the correctness and overall functionality of aggregated/orchestrated services in an open marketplaces by employing grid services).

In [63], Benatallah et al. describe the design and implementation of the *Self-Serv* system where Web services are declaratively composed using a model-driven approach, and the resulting composite services are executed in a decentralized way within a peer-to-peer and dynamic environment. Papazouglou M.P. also in [64] addresses the issues relating to the use of business transactions in Web service based applications. The paper introduces a *business transaction framework* (based on collaborating Web services)

[13] Http://www.omg.org/spec/BPMN/1.1.

that is necessary for building robust and extendible e-business applications. The author also introduce the standard Web service based initiatives such as BPEL, WS-Transaction, WS-Coordination, the Business Transaction Protocol (BTP), and the ebXML Business Process Specification Schema (BPSS) that enables the description of the public interface of e-business processes. Dalal et al. [65] evaluate the application of BTP (Business Transaction Protocol) for practical cases that has been developed by OASIS to support the collaboration of commercial Web-based applications.

Alamri et al. [66] introduce a classification of dynamic Web service composition techniques as:

- *runtime reconfiguration* using wrappers where one or more components are wrapped inside another component [67].
- *runtime service adaptation*, where the interface and behavior of services or components are changed at runtime, which proposed for incompatible service composition.
- *composition language*, which provides a way to better describe component composition. A composition language is a combination of an Architectural Description Language (ADL), a scripting language, a glue language, and a coordination language.
- *workflow-driven composition techniques*, which define an abstract process model to characterize the set of tasks and data dependencies of workflow in a composition.
- *ontology-driven service composition*, which facilitates the semantic dynamic composition of Web services.
- *declarative composition techniques* that use the composability rules to determine whether two services are composable [16].

Spanoudakis et al. [68] propose a framework to dynamically monitor composite services and dynamically identify the service which will substitute the malfunctioning services. The goal of the proposed approach is to detect and correct both functional and non-functional requirements violations. The framework can be applied on both service *infrastructure* and service *composition* layers. Skogsrud et al. [69] introduce a trust negotiation framework for Web services based on state machines, extended with security abstractions. This model-driven approach provides benefits for developers of composite Web services. Quintero at al. [70] introduce dynamic model for service composition which allow specifying the structural and dynamic requirements of Web services compositions. In [71] Casati et al. developed *eFlow* which is a system supporting dynamic composition, and management of composite services. They illustrate how the eFlow enables the specification of processes that can automatically configure themselves at run-time according to the nature and type of services available on the Internet and to the needs of each individual customer.

Barros et al. [72] propose a collection of patterns of service interactions which allow emerging Web services functionality especially those associated with behavioral interface, choreography, and orchestration. Each of the patterns captures the essence of a problem, collects references by means of

synonyms, provides real examples of the problem, and proposes solutions for implementation in terms of concrete technologies. They discuss the implementation of these patterns using BPEL. Van der Aalst et al. [73] present a survey based on some foundational concepts of service interaction. The authors also provide a set of *service interaction patterns* to illustrate the relevant challenges. The paper focuses on the main challenges of service interaction comprising *exposing services*, *replacing and refining services*, and *integrating services using adapters*.

Several approaches have been conducted for automatic or semi-automatic Web service composition. Nevertheless, today service composition is often carried out manually and this comes from these facts that 1) published services from different providers have not identical description model, 2) the number of available services proliferates exponentially in service repository, and 3) Web service composition needs to be updated at runtime with respect to up to date information from Web services.

Hull et al. [74] provide a survey on the fundamental assumptions and concepts of service composition and related issues.

2.2 Verification of Service Composition

Even though service providers prevent the publication of erroneous services, nevertheless it is crucial to verify whether a given Web service does interact properly with his partner. The composition of two services N and M is usually not foreseen at design time of N and M in advance; therefore the resulting service $N \oplus M$ may include deadlock or livelock. Accordingly, we inevitably require to analyze the correctness of $N \oplus M$. Let $N \oplus M$ be a P/T system. A minimal requirement for $N \oplus M$ to be correct is weak termination in which the final state is reachable. Numerous research studies have been concentrated on verification of the service composition model. A formal verification of composite services is necessary to guaranty the expected requirements, especially in the case where the fundamental constraints exist.

In [151] the authors apply the SPIN[14] tool for the verification of service composition. They propose the *Labeled Control Flow Graph* (LCFG) as a graphical model to represent the basic and structured activities of BPEL. For better analysis by SPIN, the LCFG is transformed into a PROMELA program to evaluate the correctness of a given BPEL. M. van Hee et al. in [75] introduce a construction mechanism based on *refinement*[15] *rules*. The proposed approach verify the weak termination of a service tree by checking pairwise composition of components: 1) refinement of a *safe* place of a component by a workflow net model, 2) refinement of synchronized pairs of places in a composition model, and 3) creation of a new component for an existing component such that the weak termination property is preserved for the overall system. The author of [76] suggests two analysis techniques for checking the properties of a composition model by *reachability tree* and *matrix equations*. Hence, a service net is live if all of the leaves in the

[14] http://spinroot.com

[15] In a service net refinement, another service is inserted into its execution path instead of one of its constituent.

associated tree are identical and represent the final marking of the net. If the reachability tree also contains the leaves different from the final marking, it would mean that there is a deadlock within the service. The proposed approach in [77] verifies the composed services by inspiring of *operating guidelines*. The approach is based on configuration process as an external service for synthesizing a partner. This technique aims to act as an improver of a service behavior despite other partners in order to remove the blocked transitions and the corresponding paths that are not used for a long times.

A fundamental challenge in verification of service composition is *compatibility* that arises from the distributed execution of heterogeneous Web services in flexible compositions. Two services are compatible if any collaboration between them is accomplished successfully, and also each of them can reach their expected results (final states). Hence, the compatibility verification is a fundamental issue for a proper interoperability among services [19][78-80].

The authors in [81][19] provide a taxonomy of compatibility for Web service compositions:

- *Interface compatibility*: it is related to the syntax and types of messages between partner services.
- *Semantic compatibility*: this type of compatibility ensures that messages and their content are correctly interpreted.
- *Behavioral compatibility*: it means there are no deadlock, livelocks, and other unconventional situations in a composed model.
- *QoS compatibility*: it focuses on some quality parameters such as time-related properties or security standards.

Despite tremendous efforts on evaluating the compatibility measures among collaborative services, there is still significant attention to it as a precondition for a correct service interaction. Indeed, the main reason for proposing the various compatibility checking approaches arises to verify effectively whether partner services can successfully interact with each other. Thus, such a requirement remains for providing an adequate technique to check the interaction behavior among services in order to validate a compatible service composition. In an appropriate method of service composition verification eventually two services can reach their expected results in addition to a proper communication. In other word, verification of the compatibility of collaborative business processes is crucial for success of the interoperability between services [82].

In the case where a service composition is not compatible, Lohmann N. [83] defines an algorithm to suggest changes of a service to achieve overall compatibility. Foster H. et al. [84] discuss a model-based approach to verify process interactions for coordinated Web service composition. The approach uses the FSM representations of Web service orchestrations. Lohmann N. et al. [85] analyze the compatibility of BPEL services and compositions of BPEL services. They provide formal semantics for BPEL4Chor choreographies which enables the application of existing formal methods to industrial service languages.

This checking includes verification of compositions with respect to compatibility and the completion of partially specified service compositions. Martens A. in [86] verifies and analyzes the compatibility of a composed system for Web service based business processes by applying the Petri nets formalism. Wu Z. et al. [80] propose an automatic method to check the behavioral compatibility in a service composition. The authors apply π-calculus to model the service behavior and interaction.

From compositional perspective, the notion of *controllability* [87] has been proposed as a criterion for analyzing the correctness of a single service net where a complete composition is usually not available at design-time. A service is called *controllable*, if there is at least one compatible partner for it. Controllability is an extension of compatibility to single services and can be seen as a fundamental *soundness* property for services in any possible composition. Weinberg D. in [88] introduces an approach for evaluating the controllability of an open net based on *interaction graph* (i.e., a model for viewing all possible communication between partner services). For a reasonable controllability analysis, the author applies some particular *reduction rules* to improve the complexity of the complete interaction graph. A full combination of an interaction graph with the reduction rules has been presented into the tool *Fiona* [50]. In the Fiona controllability is verified by trying to synthesize a compatible partner. Lohmann N. and Weinberg D. [89] introduce the tool *Wendy* which is a Petri net-based tool to synthesize the partner services. Wendy analyzes the controllability of an open net and synthesizes a partner if the net is controllable.

Xiong et al. [90] have worked on behavioral compatibility analysis of Web service interaction. They exploit a Petri net based model called *workflow module net* (WMN) for representing the service processes and their composition net (C-net). The authors perform a structural analysis of C-nets with respect to the existence of the siphon. For incompatible C-nets, they also propose the policies such as appending the additional information channels and substituting another compatible C-net. In most research efforts the authors apply the *model checking* techniques [91] to automatically verify the compatibility of service interaction. Model checking is a formal verification method that explores undesired states in a graph modeling the system behavior. There are many powerful model checkers such as NuSMV [92], BLAST [93] and SPIN [94].

In the following section, we deal with the correctness of Web service composition in particular with behavioral compatibility using structure theory of Petri nets. We also provide new way of looking at interaction services permitting us the identification of some interface patterns ensuring compatibility between two or more services.

2.3 Structural Verification of Service Composition Compatibility

In this section, we focus on the analysis and verification of Web service composition behavior. In particular, we check that neither deadlock nor livelock occurs in composition model. Usually, the

verification of such integration is achieved by using techniques based on state space exploration of a given service formal model. We here present an approach based on structure theory of Petri nets allowing the recognition of necessary and/or sufficient conditions ensuring compatible composition and a better understanding of the incompatibility sources. This part of our thesis has been published in earlier paper [31]. The fact is that verification techniques particularly structural techniques and tools developed for Petri nets can be fully exploited in the context of Web services described by BPEL [95], or others. The main goal of this approach is to show how structure theory of Petri nets can provide some guidelines and solutions for ensuring the correctness of Web services composition.

As we pointed out in the previous section, here we model Web services by means of the oWFNs. From a modeling point of view, communication between Web services takes place by exchanging messages via the interface places. Therefore, the interaction between two oWFNs is modeled by merging their respective shared elements which are the equally labeled input and output interface places. Such a fused interface place models a channel, and a token on such a place corresponds to a pending message (i.e., ready to be received or sent) in the respective channel. As it is convenient to require that all interactions are bilateral and directed (i.e., every interface place $p \in P_I$ has only one oWFN that sends a message into p and only one oWFN that receives a message from p). Thereby, oWFNs involved in a composition are pairwise *interface compatible*, i.e., only input interface places of the oWFN overlap with output interface places of the other. The interface compatibility is a basic and first requirement for services composition.

Definition 2.1 (Composable oWFNs). Let N_1 and N_2 be two oWFNs with pairwise disjoint constituents except for the interfaces. N_1 and N_2 are *composable* oWFNs if and only if they are interface compatible such that $In_1 \cap In_2 = \varnothing$ and $Out_1 \cap Out_2 = \varnothing$.

Definition 2.2 (oWFNs Composition). Let N_1 and N_2 be two interface compatible oWFNs. The interaction of two oWFNs N and M is represented by their *composition* denoted by $N \oplus M$. The *composite* net $N \oplus M$ is an oWFN where

- $P = P_N \cup P_M$;
- $T = T_N \cup T_M$;
- $F = F_N \cup F_M$;
- $In = (In_N \cup In_M) \setminus (Out_N \cup Out_M)$;
- $Out = (Out_N \cup Out_M) \setminus (In_N \cup In_M)$;
- $m_0 = m_{0N} \oplus m_{0M}$;
- $M_f = M_{fN} \oplus M_{fM}$;

The composition of oWFNs is commutative and associative, i.e., for interface compatible oWFNs N_1, N_2 and N_3, $N_1 \oplus N_2 = N_2 \oplus N_1$ and $(N_1 \oplus N_2) \oplus N_3 = N_1 \oplus (N_2 \oplus N_3)$.

Definition 2.3 (Closed net). A composite oWFN *N* with empty interface places (i.e. *In* = ∅ and *Out* = ∅) is called a *closed* net.

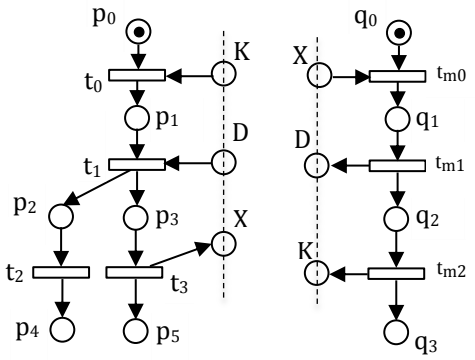

Figure 2.2. Not compatible partner services

A composite web service modeled as a closed net is a service that consists of coordination of several conceptually autonomous but interface compatible services. Although, it is not easy to specify how this coordination should behave, but we show how compatibility analysis and verification of composition of services can be efficiently undertaken using the results of the structure theory of Petri nets that we provided in the previous section. We assumed that a closed net is a workflow net. The notion of soundness has been established as a quality criterion for workflow nets implying the *boundedness* and *liveness* [96]. Our focus here is to structurally check or ensure the behavioral compatibilities for a closed net *N:*

- Weak-compatibility. A closed net *N* is said to be *weak-compatible* if and only if *N* is deadlock-free.
- Compatibility. A closed net *N* is said to be *compatible* if and only if $\forall m_f \in M_f$, m_f is home state (final state is always reachable) and *N* is quasi-live (i.e., proper termination and no dead activities). Compatibility excludes not only deadlocks but also livelocks.

Remark. Let us to precise that a deadlock state *m* in a closed net *N* is a reachable state ($\forall m_f \in M_f$; $m \neq m_f$) under which no transition is enabled. Obviously, compatibility implies weak compatibility.

Let $N = \langle N_1 \oplus N_2 \oplus \ldots \oplus N_k \rangle$ be a closed net and *Ni* be an oWFN; $N_i^* = (P_i, T_i, F_i, m_{0i}, M_{fi})$ is called the *inner subnet* of N_i. We assume that N_i^* holds boundedness property. We denote by N_i^{**} the subnet obtained from N_i^* by connecting the initial place p_i to the final place p_f via an additional transition t_i^* such that $(t_i^*)^\bullet = p_i$ and $^\bullet(t_i^*) = p_f$. We denote by θ(*N*) the net obtained by substituting the N_i^* by N_i^{**} in each N_i.

First of all, from the two well-known propositions 1.1 and 1.2, we can deduce easily the following propositions:

Proposition 2.1. *Let $N = \langle N_1 \oplus N_2 \oplus \ldots \oplus N_k \rangle$ be a closed net. If $\theta(N)$ satisfies CS-property then N is weak compatible.*

Proposition 2.2. *Let $N = \langle N_1 \oplus N_2 \oplus \ldots \oplus N_k \rangle$ is a closed net. If N is compatible then all $\theta(N)$ satisfies* CS-property (we prove that $\theta(N)$ is live).

Example. Let us consider two interface compatible oWFNs of Figure 1.3. In the corresponding closed net $\langle N1 \oplus N2 \rangle$, $\theta(N)$ satisfies the CS-property; therefore N is weak compatible, but N is not compatible. Indeed, the final marking $p_4 + q_3$ cannot be reached from the reachable marking $m^* = p_4 + K + q_3$.

Example. Now consider the closed net N obtained by composition of two oWFNs of Figure 2.2. The CS-property is not satisfied, i.e., there is an empty siphon $S = (K, X, p_1, p_3, q_1, q_2)$ at m_0, so N cannot be live neither deadlock-free. Consequently N is not compatible.

Theorem 2.1. Let $N = \langle N_1 \oplus N_2 \oplus \ldots \oplus N_k \rangle$ be a closed net. If N is compatible then all N_i^* are sound.
Proof. Suppose there is a N_i^* that is not sound (i.e. N_i^{**} is not live or not bounded).
Case (1): N_i^{**} is not live (i.e. there is a non-live transition $t \in T_i$ in N_i^{**}). As the (input) interface places only limit the behavior of the associated oWFN N_i^*, t remains non-live in $\theta(N)$, thus N cannot be compatible.
Case (2): N_i^{**} is live but not bounded, thus m_f cannot be a home state and N is not compatible.

According to previous results, the compatibility of oWFNs requires not only interface compatibility of oWFNs but also soundness of their inner subnets. We define now two classes of oWFNs namely *Ordered oWFNs* and *Root oWFNs* where soundness is equivalent to CS-property [97].

Definition 2.4. (Ordered oWFN). *Let N be an oWFN. N is called an Ordered oWFN if and only if N^{**} is an Ordered net.*

Definition 2.5 (Root oWFN). *Let N be an oWFN. N is called a Root oWFN if and only if N^{**} is a Root net.*

From these two classes of oWFNs, we define a large subclass of closed nets called *Root Closed nets* presenting realistic interfaces patterns and where compatibility can be structurally decided. In this subclass we impose a restriction on the connection nature of interface places such that root internal places are

preserved after composition (i.e. an input interface place can be a root place but it cannot take the place of another internal one). A larger subclass of composite service can be obtained by applying the basic building process of Root Closed nets in a recursive way (i.e. modules can be root closed nets or more complex nets defined in such a way).

Definition 2.6 (Root Closed net). A P/T system $N = (P, T, F)$ is called a *Root Closed net* if and only if P is disjoint union of $P1, \ldots, Pn$ and B; T is disjoint union of $T1, \ldots, Tn$; and the following conditions hold:

i) For every $i \in \{1, \ldots, n\}$, let $N_i = < N_i^{**}, In_i, Out_i >$ be an oWFN such that:

- $(In_i \cup Out_i) \subseteq B$;
- $N_i^{**} = (P_i, T_i, F_i, m_{0i}, m_{fi})$ where $F_i \subseteq (P_i \times T_i) \cup (T_i \times P_i)$ is an Ordered or Root oWFN satisfying CS-property.

ii) For every N_i^{**} ($i \in \{1, \ldots, n\}$): $\forall b \in B$, b preserves the sets of root places of N_i^{**} (i.e. $\forall t \in T_i$, $Root(t)_{Ni**} \subseteq Root(t)_{Ni}$).

iii) There is a subset $B' \subseteq B$ such that the subnet induced by the inner subnets $\subseteq N_i^{**}$ ($i \in \{1, \ldots, n\}$) and B' (denoted by $\theta(N)_{B'}$) is conservative and connected (if $B' = B$, $\theta(N)_{B'} = \theta(N)$).

Theorem 2.2 *Let N be a Root Closed net. The three following assertions are equivalent:*

- *N is deadlock free;*
- *N satisfies CS-property;*
- *N is live;*

Proof. Root Closed nets are by construction a subclass of Synchronized Dead Closed Systems (SDCS) [97], which are a K-Systems. Therefore, this equivalency holds.

Corollary 2.1. *Let N be a Root Closed net. If $\theta(N)_{B'}$ satisfies CS-property then N is weak compatible. This means that N is deadlock-free but some interface places can be unbounded.*

Corollary 2.2. *Let N be a Root Closed net such that $B' = B$. If $\theta(N)$ satisfies CS-property, then N is compatible.*

Proof. Since $B' = B$, $\theta(N)$ is live and bounded. This means that N is deadlock-free and the final marking is a home state.

In the context of Web service interaction, the approach of service *adaptation* [98-102] [22] have been realized for incompatible[16] collaborative services that have mismatches in their interfaces or behavior such that they cannot be directly composed. Significant research efforts proposed the adaptation techniques to tackle with incompatibilities of services. In the next chapter, we argue the need for

[16] We use the term "incompatibility" and "mismatch" interchangeably

adaptation of service as a typical solution for incompatible service interaction. In addition, we present an overview on proposed adaptation approaches dealing with mismatches at the level of service interfaces and business protocols. The first one refers to the mismatches between service signatures whereas the latter concerns with the messages exchanges dependencies.

Chapter 3

Adaptation of Service Composition

Despite tremendous efforts on evaluating the compatibility criteria among services [103] [31] [101] there is still considerable attention to it to verify the correctness of service collaboration. The typical approach for enabling incompatible services to interact is service *adaptation*. While standardization in Web services reduces the heterogeneity and makes their interoperability easier, adaptation still remains necessary. Adaptation functionality can be offered to enable integration inside and across enterprise boundaries. The need for adaptation in Web services comes from the following sources: *ensuring the interoperability*, *optimization*, *recovery*, and *context change*. Primarily, the mediation concept was introduced for databases [152].

The authors in [18] [101] [23] identify the needs for adaptation in Web services by addressing the heterogeneity at the levels of service *interface* and *business protocol*:

- The mismatches at service *interface* level includes service *signature* incompatibilities (e.g., message and operation name, number; the type of input/output message parameters of operations; and the parameter value constraint) with the following classifications:
  - *Syntactical*. No equality exists between service's operations name and their input/output message names. The syntactical compatibility ensures that the provided interface by a service is equivalent with the required interface of the partner and vice versa.
  - *Structural*. There are differences in the expected types or values of input/output messages.
  - *Semantical*. There are differences in the interpretation of a data element's meaning or an operation's function.
  - *Messages split / merge.* A single message of a service is matched with several messages in another service for the same functionality, or several message of a service have one matching message in another one.

- The mismatches at the *business protocol* (or service behavior) level are concerned with the message exchange dependencies among services (e.g., *deadlock* where both partner services are mutually waiting to receive some message from the other, and *unspecified reception* in which one service sends a message while the partner is not expecting it):
  - *Ordering constraint.* The constraint that services impose on message exchange sequences.
  - *Extra / missing messages*. A service delivers a message that is not specified in another service partner and vice versa.

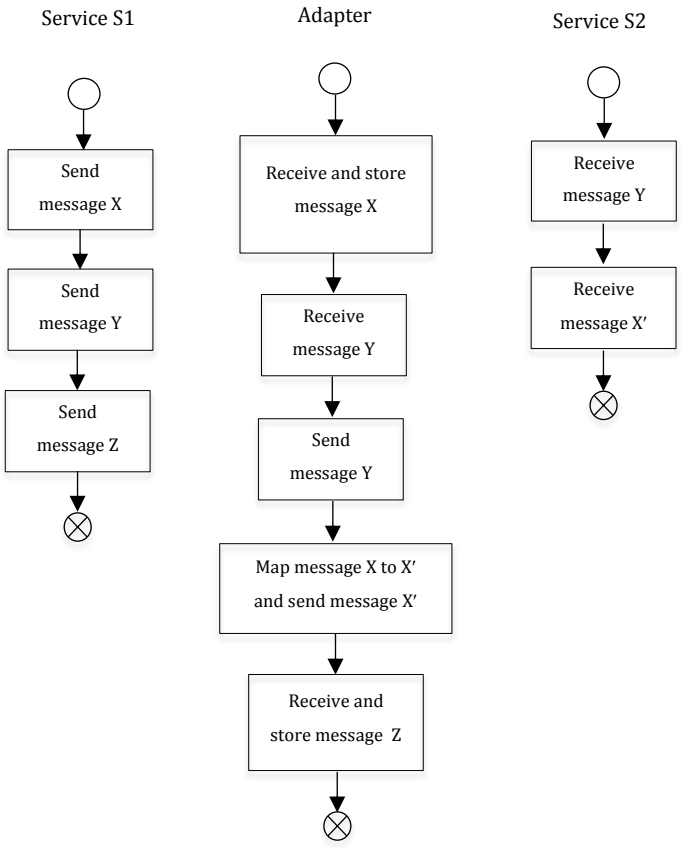

Figure 3.1. Adaptation of service composition

Numerous adaptation approaches have been proposed to tackle with both service interface and business protocol mismatches between the provided and the required functionalities of services developed by different parties [104-108] [30] [42]. The proposed approaches rely on one of these two techniques: *service modification* or *synthesizing an adapter component.*

The adaptation in terms of service modification requires applying some tuning actions to support the partner service's specifications. Whereas in service interaction where the adaptation is dealing with

creating an adapter[17], a standalone component mediates the interactions between two services with potentially different interfaces and protocols such that the interoperability is achieved. The issue of synthesizing adapters for incompatible service interaction has been studied in the area of SOA as well as in the area of component-based software engineering.

As Figure 3.1 shows, the adapter component is added between partner services that its main role is to match the messages or manage the messages exchanges. Using such a mediator two services can be correctly composed. In other words, the adapter compensates for the differences between their interfaces by *transformation functions* (e.g., XSLT, XQuery). The interface adaptation arises when the interface that a service provides does not match the interface that is expected to be provided in a given interoperability. For instance, as depicted in the Figure 3.1, service *S1* sends a message with type *X* while its partner service *S2* needs to receive the same message but with the type *X'*. Many industrial tools have been developed for interface adaptation using *schema mapping tools* (e.g., Microsoft BizTalk Mapper[18], Stylus Studio XML Mapping[19] and SAP XI Mapping Editor[20]).

Besides, the adapter reconciles the incompatibilities between service business protocols by rearranging the messages exchanges or generating a missing message [20]. In particular, some of adaptation approaches have been focused on the reconciliation of incompatibilities between *behavioral interfaces* in which interfaces capture ordering constraints between messages exchanges.

3.1 Adapter Specification and Modeling

More precisely, an adapter specification must be able to monitor and track the state of the partner services while controlling the messages exchanges between them. In addition, an adapter generally consists of a set of activities in order to perform the specific functionalities on the messages. These functionalities comprise of *receiving*, *sending*, *storing*, *synthesizing*, *merging*, *splitting*, and *converting* the messages. All of adaptation approaches propose a common logic for an adapter: *intercept a message sent by a partner service*; *store it if necessary*; *transform it to a required format supported by other partner service and/or generates a new message*; *and then send the message to the partner service that needs to receive it.*

The authors in [109] characterize the adapter functionalities based on a combination of operators such as *flow*, *gather*, *scatter*, *collapse*, *burst*, *hide*, and *create* for interface matching. Except for the hide and create operators, the others take as input a transformation function which manipulates the data in an adapter. The operators flow, gather, scatter, and hide described a syntax corresponding to the mismatch patterns identified by the adaptation approaches in [110] [23]. For instance, flow corresponds to the *one-to-*

[17] In other references, the term "adapter" is referred to "mediator" or "middleware".
[18] http://msdn.microsoft.com/en-us/library/ms943073.aspx.
[19] http://www.stylusstudio.com/.
[20] http://www.wsw-software.de/en-sap_services-mapping_sap_xi.mapping-sapxi.html.

one mapping, gather corresponds to *many-to-one* mapping, scatter corresponds to *one-to-many* mapping, and hide corresponds to *one-to-zero* mapping.

An adapter specification can be described by BPEL or FSM and/or UML activity diagram. In this thesis, we represent the adapter using the oWFN where the transitions characterize adapter functionalities and the interface places denote the messages which are arrived (or sent) from (or to) the partners.

Definition 3.1 (Adapter protocol). An adapter between two business protocols N and M denoted by $A\_\{N, M\}$ is defined as an acyclic oWFN $A\_\{N, M\} = (P^A, T^A, F^A, In^A, Out^A, m_0{}^A, M_f{}^A)$ where

- $T^A = T_I{}^A$ (i.e., $T_I{}^A$ standing for interface transitions of the adapter);
- $In^A \subseteq (Out^N \cup Out^M)$;
- $Out^A \subseteq (In^N \cup In^M)$.

As we pointed out, all messages exchanges between the partner protocols will go through the adapter. An adapter also consists of a set of typed memory cells (buffer) to store the messages received from a partner and not yet needed by the other one.

The adapter business protocol is acquired by its transitions rules to send a message to the appropriate partner, or to receive a message from a given partner, and a set of memory actions that store or retrieve messages in/from the buffer. A rule also constructs a missing message that has to be sent to the partners by adapter which can be viewed as a coordinator [20]. In the BPEL adapter [111], we assign this message to a message variable which can be allocated to the corresponding input interface place of the requester protocol. At each transition of adapter we can obtain the state of buffer at this point of coordination. Figure 3.2 shows the adapter $A\_\{P1, P2\}$ between two incompatible collaborative protocols *P1* and *P2*.

In our approach, we use the notion of $(In^A)_N$ or $(Out^A)_N$ to depict the input (output) messages that adapter receives (sends) from (to) partner N. Also, the flow relation $(In^A \times T^A)_N$ or $(T^A \times Out^A)_N$ shows that adapter A receives (sends) the input (output) message $p \in In^A$ ($\in Out^A$) at the transition $t \in T^A$ from (to) partner N.

Figure 3.2 shows a simple example of two business protocols *P1* and *P2* that are interacting via adapter $A\_\{P1, P2\}$. For instance, *P1* needs to receive message X at interface transition t_0, while *P2* submits message M, so *P1* should wait for X. Therefore, $A\_\{P1, P2\}$ stores M in its buffer until receives a request for M from *P1*. The operations in *P1* are going forward once $A\_\{P1, P2\}$ sends X to him that has previously received from *P2*. In such a way, $A\_\{P1, P2\}$ makes compatible interoperability between *P1* and *P2* by managing and re-arranging the messages exchanges.

In the next section, we review some of service adaptation approaches (i.e., either generate an adapter module or modify the specification of participant services).

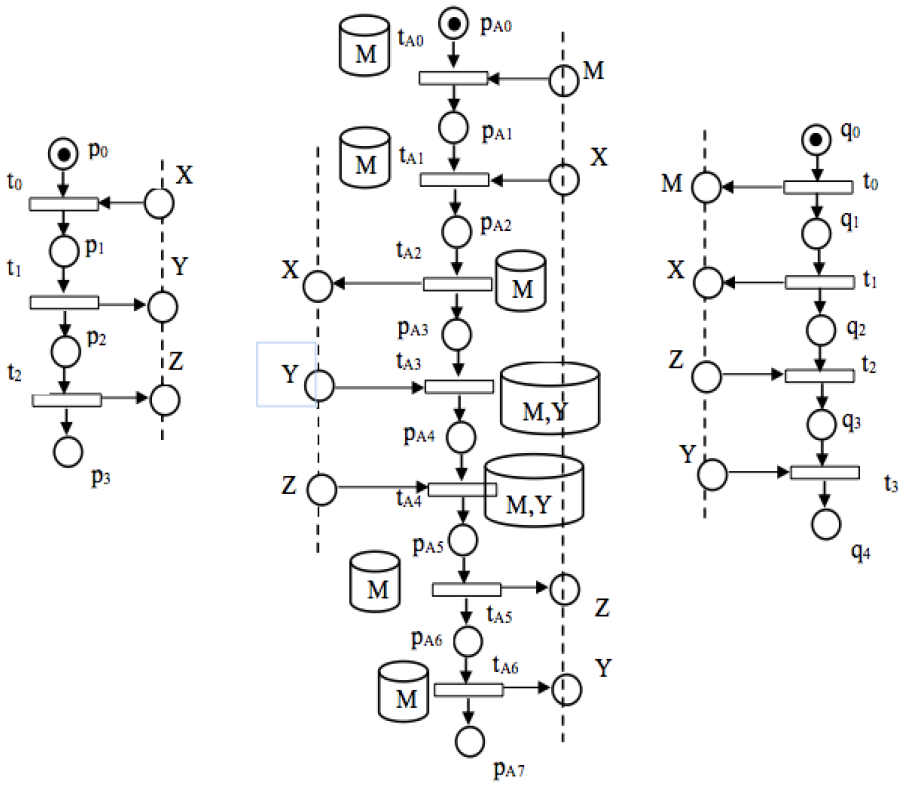

Figure 3.2. The oWFN model of adapter A_{P1, P2} (middle) between two business protocols P1 (left) and P2 (right)

3.2 State of the Art on Service Adaptation

The aim of proposed mediator in [112] is to generate the missing messages which are required to complete the Cartesian product. The authors also verify the correctness of the mediator. Sheng et al. [113] develop a middleware for the composition of Web services by focusing on the dynamic and scalable aspects of Web services composition. Baresi et al. [114] define a modeling framework for adaptive information systems based on e-services. They propose adaptation rules enabling the composition and dynamically select e-service channels according to the constraints of available architectures and application-level requirements. Deng et al. [115] provide an evaluation of the *aggregation* problem for synthesized mediators of Web services (SWMs). The aggregation problem aims to optimize the realization of a given mediator.

Li et al. [116] present an approach based on *patterns* to generate executable mediators. The authors present several basic and advance mediator patterns to reconcile all possible business protocol mismatches. They propose a heuristic technique based on message mappings to semi-automatically identify protocol mismatches when two services partially compatible (i.e., interface compatible services). The technique selects appropriate basic mediator patterns according to the identified protocol mismatches. They also

develop the corresponding BPEL templates of these patterns which can be used to generate executable mediation code.

In [117], Brogi et al. propose an approach to check whether two or more partner services involved in an interaction are compatible or not. The authors also develop the algorithms to automatically generation of the adapters. They apply the process algebra to represent Web service choreographies. Du et al. [118] present an approach to mediation-aided composition of Web services. To analyze the compatibility, the authors propose to automatically construction and analysis of the *modular reachability graph* (MRG) of composition where the problem of state space explosion is significantly alleviated. For the compatible composition, the BPEL code is automatically generated. Erradi et al. [119] present the policy-based middleware called Manageable and Adaptive Service Compositions (MASC) for dynamic monitoring and adaptation of Web services compositions. Ardissono et al. [120] present a mediation framework supporting the Web services interaction to resolve the protocol mismatches.

In the following, we review in detail some of approaches proposing adapter generation methods focusing on service interface or business protocol mismatches and/or both of them. Besides, numerous approaches presented techniques of adapter synthesis by taking the service behavior incompatibility into consideration in the interface matching.

Yellin and Strom [20] present an approach to evaluate the existence of an adapter for protocols-incompatible interfaces by applying the *execution trace*. The authors discuss two *asynchronous* and *synchronous* protocol semantics of components collaboration, and propose an automated method of adapter generation based on *synchronous* semantic.

The adapter protocol is modeled using the FSM consisting of a set of states, a set of typed *memory cells* to store the messages received by the adapter, and a set of *state transition rules*. The adapter's behavior is acquired by its transitions rules. Each rule describes a transition from a state to another in the adapter based on sending or receiving messages, along with a set of memory actions that store or retrieve messages in/from the cells. If the adapter is in a state with a send transition, the adapter may send a message to the appropriate component and enter the target state. If the adapter is in a state with a receive transition, the adapter may wait for a message to arrive and then enter the target state. A rule also constructs missing messages that have to be sent to partners. According to the synchronous semantic, when an adapter receives a message from a partner service, the adapter will not accept any message from this service until this message is forwarded by adapter and receives its response for this message, afterward, the adapter will continue to receive the other messages from this partner.

To synthesize the adapter specification for a pair of components, their interface mappings are required as the input (e.g., which messages should be mapped to which other messages). The adapter synthesis process explores all possible interactions between the participant protocols and adds them to the adapter protocol. In the case where there are states leading to deadlocks or with unspecified reception, they are removed

from the adapter protocol. The proposed adaptation approach defines *interface mapping rules* based on *parameters* rather than *message mapping*.

Taher et al. [121] present an adaptation approach based on the *Complex Event Processing* (CEP) to automatically adapt both service interface and business protocol levels mismatches. Therefore, they develop the operators for each of transformation patterns associated with the mismatch types, e.g., *match-make, split, merge, aggregate, and disaggregate.* These operators are modeled as *configurable automata* where transitions show both *observable* and *unobservable* actions. To generate a CEP adapter deployable in the CEP engine, each operator automata should be converted to a continuous query. For every message received by a CEP adapter, an input stream is created. Then an appropriate continuous query consumes the input stream and delivers the corresponding output stream for the target service. In a CEP adapter, all actions related to messages are molded as events. The CEP platform provides the conditions for running and analyzing the stream of data, and also the techniques to define the relationships between events.

Brogi et al. [111] provide a methodology for the automated generation of adapters capable of solving protocol mismatches between the BPEL processes. Given two communicating BPEL processes whose interaction may not proceed, the adaptation process automatically synthesizes a full/partial BPEL adapter (if possible). The adaptation methodology consists of *service translation*, *adapter generation*, *adapter analysis*, and *adapter development*. The proposed methodology applies the BPEL2YAWL translator [122] for transforming the BPEL processes into YAWL workflows which are used to model the service protocol. It also generates the YAWL workflow of the adapter which can be used to check the properties (e.g., deadlock-freedom, reachability, liveness, and so on) of the interaction with the adapted services. The adapter generation phase generates the Service Execution Trees (SETs) of two partner services and an adapter mediating the interaction between them. The SET of a BPEL process is a tree describing all the possible scenarios of executing the basic activities. In this phase also the YAWL workflow of the adapter A from SET(A) is generated, if the adapter has at least one successful trace.

Kongdenfha et al. [98] present an adaptation approach in terms of service modification by proposing an *aspect-oriented* paradigm. The main contribution of this approach is to apply the aspect-oriented programming paradigm to specify the adaptation template of each mismatch pattern in the form of *aspect template*. Each aspect template includes a set of (*query*, *advice*) pairs that can provide a collection of *adaptation aspects.* These adaptation aspects can be merged with *runtime instances* of an adapted service. The pair (*query*, *advice*) determines where and based on what logic the adaptation has to be applied. An *advice* defines the adaptation logic by comprising the procedures with one or more actions to resolve an associated mismatch. A *query* (query language) is described as a *join-point* where specifies in which point, a set of activities associated with the advice has to be executed. The aspect template for interface mismatch can be initiated by providing the *query input parameters* and XQuery / XSLT transformation function as *advice input parameters*. In the case of a business protocol mismatch, if a service needs to receive an *ack*

message, while the partner does not supply it, the adapter can provide such a message by evaluating the list of older messages exchanges.

Moreover, the authors argue their motivation to exploit the aspect-oriented methodology for dynamic plugging and unplugging the adaptation logics to partner services at *runtime*. They also try to provide conditions for deciding where each of two methods, services modification (respecting aspect-oriented methods) and generating stand-alone adapter, is preferable. In addition, they implement a tool that assists the developers in semi-automatic generation of their adapter logic. Although, these authors in [108] present an approach by focusing on both adaptation methods (i.e., *developing standalone adapter* and *service modification*). They introduce *mismatch patterns* for service interface and business protocol mismatches. The proposed mismatch patterns include *adaptation logic templates* which can be initiated by developers to generate adapters for resolving the associated mismatches.

Benatallah et al. [23] present a methodology for developing adapters based on *mismatch patterns*, which are used to reconcile the possible differences between collaborative services. The proposed approach provides the *adapter template*s specified in BPEL code for each mismatch pattern scenario. Indeed, they characterize different kinds of adapter templates as a solution to resolve associated mismatch which is identified among partner services. Reusing these adapter templates help the developers to synthesize their stand-alone adapters.

Hung et al. [123] present an adaptation approach for *synchronous* service interaction. This approach analyzes only the mismatches at business protocols level assuming that the service interface mismatches are resolved. The proposed framework represents Web services in the form of IA4WS (Interface Automata for Web Services) which is a customized model from Interface Automata [124]. The approach applies the idea of *one-session service* to avoid the *unwanted traces* by dealing with *unbounded messages* (i.e. which may modify the required functionality of services). The proposed adaptation approach includes (1) evaluating the interface level compatibility of services; (2) applying the model-checking tool SPIN to analyze the business protocol compatibility; (3) analyzing the need for adapter generation; (4) applying a *pushdown* model-checking for verification of adapter in terms of two specifications *fairness property of looped transition* (unbounded messages), and *fairness property of branching transitions*.

A pushdown automaton namely Interface Pushdown System (IPS) represents the adapter model. Within the IPS adapter, there are three kinds of transition, e.g., *push transition* for messages reception; *pop transition* for message delivery; and *internal transition* that is not related to messages exchanges. The adapter model never creates a message by itself. Also, applying model-checking technique for verification of adapter during its generation saves more time and cost compared with the cases where service adaptation and verification are performed separately.

Tan et al. [101] present an automatic approach of service composition and formally checking the compatibility of two collaborative services respecting the *asynchronous* semantics. They also propose an adapter generation algorithm, if two services are partially compatible. The partial compatibility implies two

services are functionality compatible, but due to incompatibility in their interfaces or business protocols, direct composition is not possible.

The proposed adaptation approach applies a *state-space* method where introduces the concept of *Communication Reachability Graph* (CRG) which concurrently constructs the *reachability graph* of two collaborative services using *data mapping*. The CRG is then verified to evaluate whether the adapter generation is necessary. The CRGs are used as a reduced state-space of the composite service. The authors discuss the incompatibility at the level of service interfaces. If the direct composition between interfaces is not possible, a request for data mapping is delivered to build the CRG to verify the adapter generation. The data mapping defines the rules as <*src*, *target*, *trans-flag*> to relate (syntactically, semantically equivalent) elements of two messages that are from two interfaces belonging to different services. For a direct map from *src* to *target*, the *trans_flag* Boolean variable is set to be *false* and otherwise is set to be *true* for additional transformation. In this approach, BPEL services are transformed into Colored Petri Nets.

Mateescu et al. [99] propose an adaptation technique which reduces the computational complexity of adapter generation using *on-the-fly exploration* and *reduction technique* (i.e., avoids full state space generation of adapter structure). The adaptation method covers both interface and behavioral mismatches of services. In this approach, service interface is modeled by the WSDL and its protocol is represented by means of *Symbolic Transition Systems* (STS).

The proposed approach takes the parameters coming with a message into consideration in the interface matching. The adaptation approach first constructs the *adaptation contracts* which describe how mismatches can be resolved. The service interfaces and their adaptation contracts (interaction constraints specifications) are then encoded. Accordingly, the LOTOS code is generated. The LOTOS encoding allows the automatic generation of adaptor protocols. Moreover, LOTOS encoding enables the adapter verification using model-checking tools.

In this framework, applying on-the-fly algorithms increases the efficiency of adapter generation. The reduction of adapter is performed by removing all actions that are not related to the service business protocol (e.g., the internal transitions, and all interactions with store activity). The authors also apply a model-checking tool to verify the generated adapter. The verification tool implements two operations: (i) *deletes the states leading eventually to deadlocks*; (ii) *keeps only the states leading to the transitions labeled by a given action (here Final)*. The adaptation contracts are built by designer, hence additional adapter verification are needed to implement several static evaluations for verifying that the adaptation contract is correct. The adapter behavior also modeled in STS.

Dumas et al. [125] discuss the service adaptation at service *interface* level, and propose an approach to reconcile mismatches between *protocols-incompatible* interfaces. The authors present a *declarative* approach for an *adapter execution engine* rather than using programming language in order to reduce development and maintenance cost.

In this approach, service interface adaptation is accomplished via mapping operators such as *flaw*, *scatter*, *collapse*, *burst*, and *hide* based on common mismatch patterns. Therefore, they propose a *visual notation* for mapping the interfaces using *algebraic* expression of transformation operators by focusing on *behavioral* aspects of interfaces. The behavioral interfaces are defined in terms of *communication action schema* using the UML activity diagram where actions are named according to the type of sent or received messages. The *deadlock* situation (i.e., message ordering constraint in interface) is verified by defining the condition on interface mapping expression, i.e. *"it is not possible to send or receive the information that is dependent on other information that we have not yet sent or received"*. The service *mediation engine* also supports the *visual notation*. Mediation engine is modeled by the FSM for an abstract representation of behavioral interfaces. The interface mappings are implemented in the mediation engine based on the logic of transformation operators. The mediation engine comprises of an *administrative console* to monitor the histories of all actions carried out by the engine. Besides, the mediation engine can store *unbounded* number of messages.

Motahari-Nezhad et al. [22] propose a semi-automatic adaptation approach for service interface matching by taking the protocol specifications into considering. In terms of interface matching, the authors introduce a method to identify message split/merge mismatch type where some elements of a message $m \in I_s$ (I_c) are matched with elements of $m' \in I_c$ (I_s) and some other elements of m are matched with some elements in $m'' \in I_c$ (I_s). The authors also present the protocol-based interface matching algorithms (i.e., *depth-based interface matching*, and *iterative reference-based interface matching*). In a depth-based interface matching approach, messages with the same or similar depths in two protocols may have a higher chance of matching, and also the infinite loops are avoided. Iterative reference-based interface matching approach includes the knowledge of previous matching (i.e., a pair of messages is selected as the best candidate match in each iteration and this pair is referred to as a reference pair for the next iteration. Potential deadlocks can be avoided by penalizing the conflicting match pair).

Seguel et al. [126] illustrate an automated approach of adapter generation to resolve protocol mismatches not for all interaction but only for a minimal set of messages exchanged that needs the adaptation. The authors analyze service collaboration from *synchronous* semantic standpoint.

The BPEL representation of a service protocol is modeled by a *protocol tree*. The leaves of this tree correspond to the basic activities and the internal nodes represent the structured activities. The proposed approach exploits the *behavioral relation* (*Seq*, *XOR*, *AND*) of the protocol syntax (i.e., structure of the protocol tree) to identify the mismatches, and recognizes the set of message exchanged that have to be adapted. To this end, an Interaction Analysis Matrix (IAM) is built over the pairs of interaction for comparing and assessing the behavioral relation of their nodes. Accordingly, they demonstrate an algorithm that can address the minimal set of interactions needing adaptation. Indeed, an IAM is able to evaluate which parts of collaboration have to be adapted, which parts cannot be resolved. Therefore, the adapter generated through this approach has lower complexity and improved performance characteristic at runtime.

Shan et al. [127] present a technique for a flexible, on-the-fly adapter generation based on *message* and *control flow* adaptation. This approach can also combine control flow constraints into message adaptation. The main idea of their framework is to provide *message transformation patterns* which can be extended for user's future requirements. Therefore, they first characterize the types of message mismatches, and then propose a set of patterns for resolving them. These patterns are defined as *message data generation*, *message data casting*, *message fragmentation and aggregation*, and *message reconstruction.* The proposed approach also exploits a set of standard patterns for control flow adaptation (e.g., *sequence, parallel, choice* and *loop*). In addition, a *compatibility matrix* based on these patterns is represented as a guideline for processes incorporating. The authors also illustrate an adapter generation algorithm.

Mooij and Voorhoeve [100] present an automated approach for adapter generation. The authors develop a proof technique for partial adapters based on the open Petri-net [128] formalism. They apply this technique to the adapter generation approach from [129] where behavioral adapters is proposed to adjust the communication between two services such that certain behavioral properties (e.g., deadlock-freedom, or weak termination) preserve in the composite service. The authors particularly address concepts like *operating guidelines*, *controllability*, *accordance*, and *partial adapters*. Their proposed approach relies on this point that the adapters are as controllers for the disjoint composition of two partner services.

As pointed out, the authors in [129] present an automated adaptation approach (using Petri net formalism) that synthesizes the behavioral adapters. They propose a specification of elementary activities (SEA) consisting of a set of transformation rules on message types. Therefore, the adapter specification includes the given partners services and the SEA. In addition, their generated adapters do not need to be verified, since the synthesis of the adapter guarantees the adapters correctness by construction.

Motahari-Nezhad et al. [130] present an approach to semi-automatically identify the split/merge type of interface mismatches, and automatically recognize the protocol mismatches by generating a *mismatch tree*. Besides, they propose an *adapter simulation algorithm* which explores all possible messages exchanges between two service interfaces to evaluate the respective protocol mismatches. The algorithm results in an *adapter protocol*, and a *message tree* that represents all deadlock situations between two service protocols. The interface matching is done by finding the match between *data elements* of the XML *schemas* associated to service interfaces. To increase the matching accuracy, they suggest the inclusion of message name and message type in an operation definition.

Furthermore, the authors discuss some *evidences* that can be used to identify (create) messages in common deadlocks:

- Evaluating the relation between data structure of message *m* (engaged in deadlock and need to be identified) and all of messages of the same interface that has been received before the deadlock point in the adapter. Indeed, if the elements of message *m* could be matched with the elements of any of these messages, there is the possibility of creating the message *m*.
- Applying the *previous interactions' logs*.

– Providing *the adapter developer's input* to identify the missing message in a deadlock and the input parameters of the transformation function.

In this work, the service business protocols are modeled in the form of FSM. They also argue that for a given interface mapping, it is necessary to take the service protocol incompatibility into account.

Wang et al. [131] characterize a *runtime adaptation* approach for service *behavioral interface* incompatibilities. To this end, they introduce a *service adaptation machine* that includes a repository of *mapping rules* where each rule is associated with an *adaptation scenario*. When a directed single mapping rule cannot be applied, the adaptation machine exploits *backward-chaining* algorithms for firing the rules to detect the behavioral mismatches, i.e., *deadlock* and *information loss* (*unspecified message*).

The adaptation idea of this paper has been implemented as a tool namely *Service Mediation Engine* (Megine). The Megine manages a number of adapters associated with specific pairs of service instances by using the *identifiers* and *references* of the received messages. In addition, the Megine through an *administrative console* is able to implement an adaptation scenario where before forwarding a message by adapter to a partner service, it is possible to check whether the target service is in a given state to consume that message. In this approach, both the partner service interface and adapter protocol are modeled using the FSM specifications.

Wohlstadter et al. [132] present the policy-based programming model, architecture, and details of their proposed middleware called. They also discuss new challenges like distribution of middleware services that arise from such a context. The *Cumulus* middleware can be dynamically customized to support diverse Web services interoperability protocols that the applications need to engage in. The authors aim to find an appropriate programming model to *describe*, *discover*, and *compose* middleware as separate services (MW services). Besides, the technical details of the *Cumulus4BPEL* prototype are provided which developed specifically to support runtime interoperability in dynamic SOAs.

Becker et al. [133] present an engineering approach to component adaptation which is based upon introducing a taxonomy of component mismatches, and identifying a number of adaptation patterns to be used as generic and systematic solutions for eliminating them. The provided taxonomy summarizes the different types of component mismatches into categories, and classifies them according to a hierarchy of *interface models*. Each of the distinguished interface models determines a set of properties which belongs to a component interface as mentioned in the following:

– The *syntax-based interface model*, which focuses on *signatures* as constituent elements of component interfaces, supports the identification and elimination of signature mismatches.
– The *behavioral interface model* also contains assertions (i.e. pre- and post-conditions) for the operations, which have been declared in the required and provided interfaces. By making use of an interaction-based interface model which focuses on describing the interaction that takes place between connected components in the form of message calls, developers are able to diagnose and eliminate *protocol mismatches* (i.e. ordering and extra/missing of messages).

- The *quality-based interface model*, which focuses on describing the Quality of Service (QoS) (i.e., which is being provided by each of the interface operations by describing a set of quality attributes). By making use of an interface model that is based on the ISO 9126 quality model, it is possible to detect and eliminate the quality attribute mismatches such as *security*, *persistency*, *transactions*, *reliability*, and *efficiency*.
- A *conceptual* interface model, which describes the conceptual semantics of component interfaces as an ontology supports the identification and elimination of so-called *concept mismatches*.

The presented approach demonstrates both functional and non-functional adaptation patterns.

Bucchiarone et al. [134] propose an extension of a basic iterative service-based applications (SBA) lifecycle along with the elements to deal with the adaptation-specific needs. From point of view of the authors, adaptation works properly only in the case the application is designed to be adaptable, so they focus on the design phase, and suggest a number of design principles and suitable guidelines to enable adaptation. For the same SBA, several adaptation strategies can be adopted and the selection of the most suitable one can be a complex issue, hence multiple criteria have to be considered. Therefore, they provide the guidelines to support this selection. The authors also discuss the effectiveness of the proposed methodology by means of the real-world scenarios over various types of SBAs. The paper aims at overcoming the fragmentation in current approaches for SBA adaptation.

Ardagna et al. [135] developed a general framework for flexible and adaptive execution of managed service-based processes. The framework, PAWS (Processes with Adaptive Web Services), coherently supports both process design and execution. The PAWS provides methods and a toolset to support design-time specification of all information required for automatic runtime adaptation of processes according to dynamically changing user preferences and context. The authors focus on how PAWS selects and adapts candidate services for a composed process.

In PAWS, both service discovery and service selection are driven by both functional and non-functional aspects. From a functional perspective, PAWS includes a *mediator configurator* to support set-up of the related runtime module – the *mediator engine* (i.e., to translate between the two service-interface signatures). If it's not possible to automatically derive message transformations, the designer defines them during process execution by providing additional information about parameter and service mappings. Their proposed mediation engine support service invocation, dynamically binding a generic candidate service without requiring stub compilation at design time, and manage service substitution which might involve services that are described by different signatures but have the same choreography.

Whenever the BPEL engine invokes a task, the mediator selects the first service from the optimizer's ranked candidate-services list. If the candidate service's interface differs from the interface that the task definition requires, the mediator retrieves the proper mapping document produced by the mediator

configurator, and then invokes the candidate service by sending transformed messages. The mediator manages candidate-service invocation through sessions.

Cavallaro et al. [136] identify a number of possible interface or protocol level mismatches between the interacting partners in a dynamic service composition. The authors address basic mapping functions for solving the simple mismatches. In addition, they propose to combine such mapping functions in a *script* (defined in a domain specific language) to solve complex mismatches. Both simple functions and scripts are executed by a *mediator* to perform the required adaptations.

The script language is composed of the *rules* structured in two parts: *mismatch definition part* that specifies the type of the mismatch to be solved by the rule; and *mapping function part* that contains the name of the function to be used to solve the mismatch. The rules can characterize the solutions for protocol or interface mismatches.

Denaro et al. [137] present an approach to design self-adaptive service-oriented architectures. The authors propose a self-adaptive solution that enables the client applications to easily and automatically adapt their behavior to the alternative Web services providing compatible functionality through different business protocols. The proposed framework automatically synthesizes the models which approximate the interaction protocols by tracing the successful interactions of Web services. It then dynamically handles the adaptations of client. On this concern, the approach adds an Interaction Protocol Service Extension (IPSE) to the SOA such that each Web service is associated with an IPSE that serves as a proxy for the Web service. Before forwarding the requests to the Web service, the IPSE monitors the sequences of interactions between the client and the Web service. The IPSE uses this information to incrementally synthesize a model that infers the interaction protocol of the Web service. The main limitation is manual testing by the developers to analyze the candidates.

Jiang et al. [138] propose an approach for adaptation of two or more services. The authors introduce new model called *protocol structure* to characterize the service business protocol using message and message dependencies. They also provide formal definitions for mismatches. The proposed approach covers various forms of service interactions such as *one-to-zero*, *one-to-one*, *one-to-many*, *zero-to-one*, *many-to-one* exchange of messages, and *message broadcast*. Besides, the approach can automatically detect multiple mismatches at a time and automatically resolve them. The authors present a simple visual tool for automatic adaptation of services and generation of BPEL adapters. The process of developing an adapter including:

1. *Initial adaptation set* (*initial* message dependency relation of adapter) is semi-automatically generated based on *interface mapping* among services.
2. Creation of protocol structure of adapted composition of two services using the initial adaptation set.
3. *Final* and *correct* adaptation set among services is obtained through automatic detection and resolution of message ordering mismatches in the protocol structure (i.e., using an algorithm to compute dead configuration):

- Extra messages are received and stored in the buffers by adapter;
- New messages are provided for missing messages (deadlock).

4. Automatically generation of BPEL adapter based on final adaptation set.

Chapter 4

Web Services Adaptation in the presence of Business Protocol Evolution

Obviously, compatibility analysis [139] and adaptation techniques to enforce service interoperability are not new ideas. In previous chapter, we have reviewed various adaptation approaches proposed to encounter the incompatibilities in Web service collaboration. The main challenge of this dissertation is to concentrate on service adaptation especially for the cases in which the business protocols of services engaged in interoperability evolve. In all associated works on service adaptation is assumed the business protocol does not change. However, there are two cases where service business protocols change during the interactions of services: one is that most of the real world services provide interaction patterns for a group of related activities such that in one interaction session, two services may engage in conversations that include two or more of such interaction patterns. One example of such services is *Google Checkout*[21]. The other scenario is when the business protocols of a service evolve.

The main question is what happens to adapter development in these cases. Indeed, our contribution is to propose an efficient solution for service adaptation in the presence of business protocol evolution. In other words, we provide an evaluation approach to decide how an adaptation mechanism between two services must be carried out in an efficient manner when one or both of them migrate to the new version through business protocol evolution.

Obviously, building a new adapter from scratch whenever the business protocol evolves sometimes is expensive. Therefore, we prefer approaches to update the adapter specification only for parts of protocols that have modified rather than totally state space exploration of the composed model (i.e., adaptation from scratch), if it makes sense. Consequently, to achieve a reasonable answer for such ambiguity issues we aim to more focus on them. Therefore, we are concerned with providing a technique to evaluate automatically,

[21] http://code.google.com/apis/checkout/

which extent of a service business protocol has impressed, or which extent remains without any change and subsequently which extent of an interaction have to be handled for adapting with the changes. Accordingly, our method has to determine how adaptation mechanism must be re-carried out for those affected interactions with the new version of service protocols. This technique must consider those elements of an adapter protocol that need to be re-managed after happening these changes. However, in some cases of business protocol evolution is more efficient to accomplish the adapter generation from scratch.

In this dissertation, we aim to challenge the service adaptation to cope with the changes in services particularly their business protocols evolution.

4.1 Web Service Evolution and Service Interaction

For variety of reasons, service providers needs to improve their services capabilities, for instance to achieve high productivity. The evolution of Web services is inevitable because the clients will always ask for new features. In fact, the existing Web Services require to be changed to adapt to the new business demands. The main issue through such amelioration is to necessarily evaluate whether the correctness of the new interactions with the new version of services can be verified.

The authors in [140] [24] present a theoretical framework to control and manage the evolution of services. They discuss Web service evolution from external observable aspects of a service including:

1- *Structural changes*: affect the service signature such as messages, operations, and data types.
2- *Behavioral changes*: focus on changes related to the business protocol of services. Indeed, the business protocol of services change due to changes in the policies, regulations, and changes in the operational behavior of services.
3- *Policy induced changes*: include the changes of Quality of Services (QoS) properties.

Moreover, Z. Feng et al. [24] represent a classification of service evolution from different perspectives:

- *Perfective* Motivation:
    1- Enhance/replace existing functionality of services,
    2- Reduce existing functionality services,
    3- Business rules change.
- *Corrective* Motivation:
    1- Service interface changed,
    2- Interface semantics modified, and
    3- Service policy is changed.

Related to this space, the focus of this work is the evolution of services behavior (i.e., business protocol changes). The authors in [141] present an approach to modify the client protocol in a static service evolution (structural changes) to support the service interaction with regard to the compatibility notions.

They introduce a set of elementary operations to model the changes of service protocols (e.g., the operators to add/remove states and transitions; and to change the initial and final states).

When a service evolve, the authors in [142] suggest a method for updating the client interface respecting the compatibility criteria. They assume that services are as black-boxes which cannot change. Their proposed approach detects the changes by comparing the interfaces of an upgraded service with its client. Also, the behavioral mismatches are identified. Besides, the compatibility measure between the updated client and related service is evaluated. S. Rinderle et al [143] discuss how the changes of a service private process may influence the associated public view, and also the public and the private processes of its partners. They propose an adaptation technique for automatically propagating such changes to the partner processes.

S.H. Ryu et al [144] investigate the problem of service protocol changes for a dynamic service interaction where multiple instances of a service protocol are running through different clients. Their main challenge is to manage such a protocol evolution regarding its running active instances. The presented approach in [145] also analyzes the evolution of services from point of view of service dependencies. The internal relation between the constituents of each service is calculated, and then the relation of two services' elements is detected. Consequently, the impact of change in a given element of a service is assessed. The authors in [146] propose a technique called *chain of adapters* to manage the evolution of service interface to preserve the compatibility with the client.

The problem of service adaptation has been investigated extensively in the literature. These approaches cover mismatch identification and adapter generation both at the interface-level and also business protocol levels. However, none of these approaches consider the issue of adapter evolution in the presence of changes in the business protocols participating in the adaptation. In our work, we go beyond the existing literature by identifying possible evolution patterns in business protocols of services, proposing algorithms to automatically identify the impact of changes in terms of the occurrence of any of these patterns in an evolved business protocol with respect of an existing adapter, and a method to automatically identify whether the adapter specification can be updated on-the-fly to remedy the changes. This will save times and efforts in assessing the impact of changes and regenerating the adapters from scratch with every change.

Our proposed approach of evolution-aware analysis in this thesis is different from current approaches wherein propagate the changes of service protocol towards their client protocols to preserve the compatibility.

4.2 Dynamic Adapter Re-configuration

In this section, we aim to present the main contribution of this thesis in which we challenge the adapters' adaptation to comply with the evolution of service business protocols. Indeed, we intend to evaluate and comprehend the impact of business protocol evolution on the adapter specification. We

present an automatic approach to investigate when it is possible to dynamically update the adapter specification without the need for re-generation and re-deployment of adapter.

To this end, we characterize the potential impacts of the changes arising from the evolution of service business protocol on the adapter specification. We present an algorithm to automatically figure out the impact of evolution on the adapter for common patterns of evolution in business protocols. The impact analysis algorithm enables us to detect the circumstances in which the current adapter can be updated on-the-fly.

We developed a prototype tool implementing the proposed approach. The experimental evaluations show a considerable saving in time, effort and cost compared to the static adapter generation methods. Experiments also demonstrate that the proposed approach significantly assists the developers to recognize when they can avoid generating the adapter from scratch.

In the next section, we first provide a motivating example.

4.2.1 Motivating Example

In this section, we present a real-world scenario (i.e., based on RosettaNet[22] standard) as a motivating example for better illustration of our proposed approach. Figure 4.1 shows the scenario of a *purchase order. Buyer orders products from Seller. Seller accepts Order and confirms order fulfilment. Seller sends Advanced Shipment Notification to buyer. Seller then sends Invoice to Buyer. Buyer may cancel the order or confirm the Invoice. Following Invoice confirmation, Buyer pays Invoice. Seller may receive order cancelation or payment. The process ends with delivery of the order to the Buyer.*

Figure 4.2 depicts the business protocols of *Buyer* and *Seller* denoted by *Q* and *P* respectively in the form of oWFN. As illustrated in the Figure 4.3 an adapter component *A_{Q, P}* is needed to support the interaction between *Q* and *P*. Indeed, due to the protocol mismatches between *Q* and *P* (i.e., in terms of unspecified reception and messages order constraints) direct composition is no possible. For instance, *Buyer* confirms invoice and sends InvoiceConf, while there is no need in *Seller* to receive this message, thus adapter *A_{Q, P}* stores it in the buffer at the level of transition tA_{12}. In addition, *Buyer* first submits the output message BuyerInfo and then receives shipment notification by the input message ShipNotif. Whereas *Seller* first sends the message ShipNotif and then receives BuyerInfo. Consequently, as shown in the Figure 4.3 the adapter should manage the sequence of such messages exchanges.

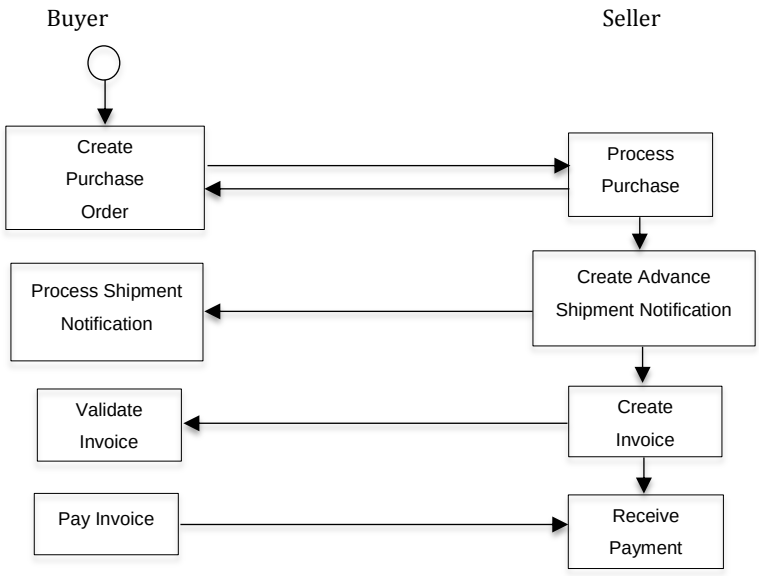

Figure 4.1. The partner services Buyer and Seller in a purchase order scenario

[22] http://www.rosettanet.org

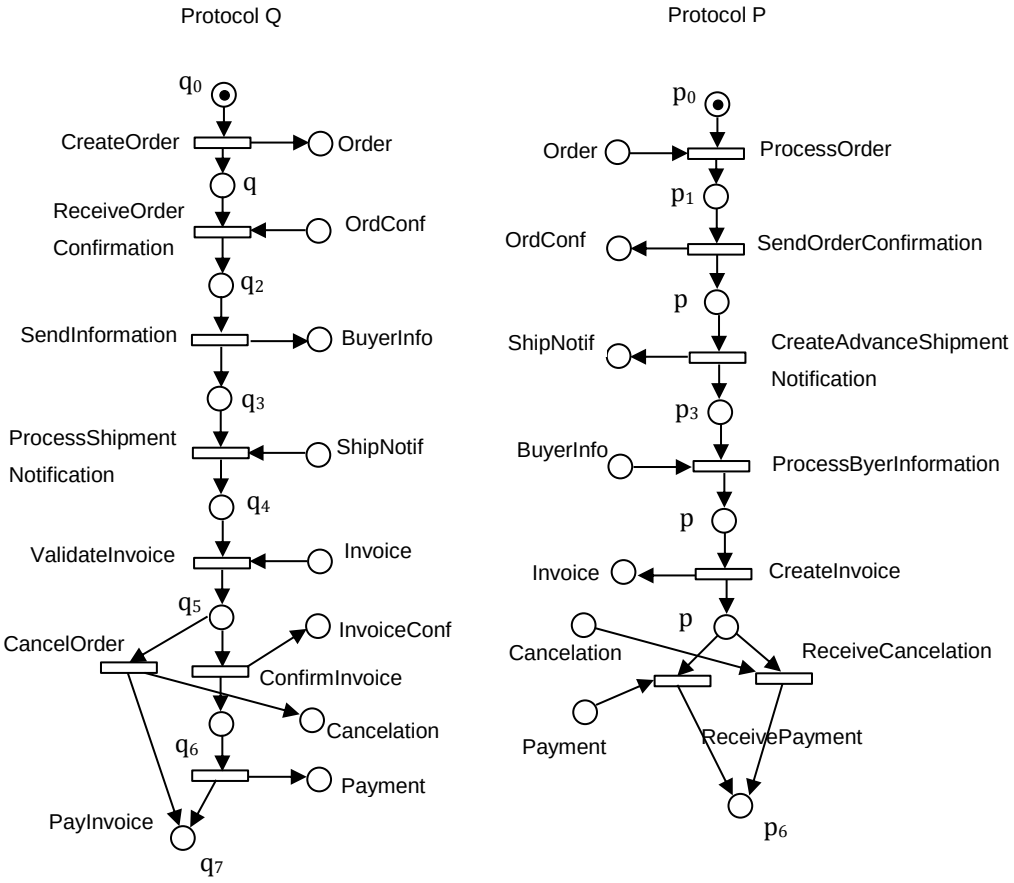

Figure 4.2. The business protocols of *Buyer* (Q) and *Seller* (P) in the form of oWFNs

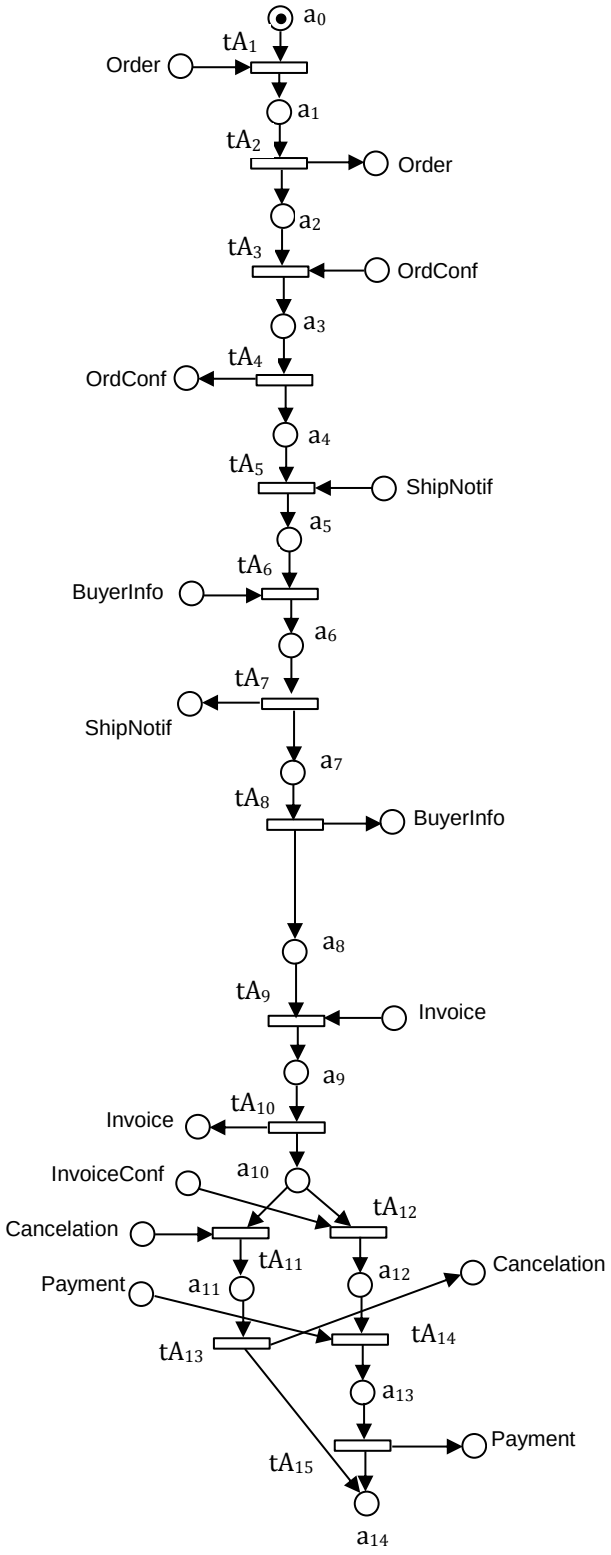

Figure 4.3. The model of adapter that sits between partner protocols *Buyer* and *Seller* of Figure 4.2

4.2.2 Business protocol evolution patterns

In this section, we define the common *adapter adaptation patterns* (AAPs) for business protocols evolution, and we also propose a method to identify these patterns.

4.2.2.1 Adapter adaptation patterns

Similar to the *design patterns* in software engineering [147], we define the *adapter adaptation patterns* (AAPs) to characterize the business protocols evolution, their respective impacts on the current adapter, and a solution showing how to fix dynamically the adapter, if possible. More precisely, the structure of the AAP consists of :

- *evolution pattern* – to capture the possible changes on business protocols evolution;
- *potential impact* – to describe the effects of respective change on the current adapter;
- *adaptation solution* – to define the operations to dynamically re-configure the specification of current adapter.

The structure of an evolution pattern defines the type of an elementary change corresponding to the occurrence of the added or removed messages. In terms of oWFNs, these changes are translated into four patterns, which are added (or removed) input (or output) interface places. Moreover, the occurrence of changes at the level of activities is described as added, removed, or updated interface transitions (i.e., AT_I, RT_I, or UT_I, respectively) that can easily be deduced in terms of these four evolution patterns.

For each of the AAPs, the potential impact of an evolution pattern is evaluated respecting the specification of current adapter. We analyze the impact of the evolution patterns from two standpoints: *partial* and *global* impact. In the case where the impact of an evolution pattern on current adapter is partial, definitely allows updating the adapter on-the-fly. For the evolution pattern with global impact, we are forced to re-generate the adapter from scratch at design time through a complete analysis of the new interactions among participant protocols. We also establish a solution of dynamic updating the adapter for the evolution patterns with partial impact. Such a method either is to add a patch – to extend the adapter from a given transition, or to change the adapter – to fix the corresponding transition.

We suppose that only the protocol P is changed (we say here P becomes P'). The challenge is now from point of view of P to evaluate the impact of changes (of its messages or activities) on $A\_\{Q, P\}$, and to dynamically conceive an appropriate $A\_\{Q, P'\}$ between Q and P' in the case of partial impact. For both impact analysis and dynamic adaptation, we denote by $\mathbb{B}(t)$ the buffer of current adapter $A\_\{Q, P\}$ at the level of transition t. To dynamically perform the re-configuration of current adapter, we also need to identify the transition t_A of adapter $A\_\{Q, P\}$ corresponding to each updated or removed transition t of protocol P (denoted by $t_A \leftrightarrow t$). The identification of t_A is easily obtained via the input or output interface

place of *t* in *P*. We denoted by *A_CrT*, the set of transitions of current adapter associated with all updated or removed interface transitions of a business protocol. We illustrate a classification of AAPs as follows:

AAP #1 – Input Addition Pattern.

- *Evolution pattern*: *input interface place is added*. This pattern deals with the case where the input interface place *ip* is added to the interface transition *t* of protocol *P′*. Indeed, the transition *t* is either a new interface transition receiving the input place *ip* ($t \in AT_I^{P'}$), or *t* is an existing interface transition in which the input (or output) interface place is replaced by *ip* ($t \in UT_I^{P'}$).
- *Potential impact*. To identify the impact of the evolution pattern on current adapter, we analyze the specification of adapter *A_{Q, P}* as follows:
  - The message *ip* exists in the buffer of *A_{Q, P}* (i.e., cond1. of Table 1). Therefore, the impact of pattern on *A_{Q, P}* would be partial. For example, suppose the *Seller* via *P′* receives the message InvoiceConf by a new interface transition ReceiveInvoiceConfirmation prior to the transition ReceivePayment).
    - *Adaptation solution.* To dynamically update the *A_{Q, P}*, we first check whether the buffer $\mathbb{B}$ includes *ip* at the transition t_{AX} of *A_{Q, P}* corresponding to *t*, i.e., $ip \in \mathbb{B}(t_{AX})$. Otherwise, we detect the transition t_{AY} with the largest prefix number in which $\#t_{AY} < \#t_{AX}$ and $\mathbb{B}(t_{AY})$ contains *ip*. Accordingly, *A_{Q, P}* is extended on-the-fly at t_{AX} or t_{AY} to send *ip* into *t*. In the case where the transition *t* is a new interface transition such that no transition t_{AX} exists in *A_{Q, P}* corresponding to it, *A_{Q, P}* is dynamically extended at the transition t_{AZ} where $\mathbb{B}$ includes *ip*.
  - The buffer $\mathbb{B}$ does not include the message *ip* (i.e., cond2. of Table 4.1). Therefore, due to a potential deadlock, the impact of the evolution pattern on *A_{Q, P}* is global. For example, suppose *P′* needs to receive the message Order&BuyerInfo instead of Order at the transition ProcessOrder. Although, the evolution of *Q* in the future may satisfy this request of *P′*, but we assume that the probability of such an evolution of *Q* during Δt is low (i.e., Δt is the total time of both impact analysis and applying dynamic adaptation on *A_{Q, P}* for *P′*). In the case where *A_{Q, P}* is able to generate *ip* (i.e., using some *evidences*, or creating *mock-up* message), the impact of pattern on adapter would be partial, (i.e., cond3. of Table 1 where the set of output interface places of *A_{Q, P}* includes *ip*).

Table 4.1. The specification of AAP #1.

Evolution Pattern: ($\exists$ t $\in$ $T_I^{P\prime}$ $\wedge$ $\exists$ ip $\in$ $In^{P\prime}$) $\ni$ ip $\in$ $^\bullet$t is a new input place

Potential Impact:

Con1. ip $\in$ $\mathbb{B}$ ($\exists$ t_{AZ} $\in$ T_I^A $\ni$ ip $\in$ $\mathbb{B}$(t_{AZ})) $\rightarrow$ Partial impact $\rightarrow$AdaptSolution_1a (t_{AZ})

Con2. ip $\notin$ $\mathbb{B}$ $\rightarrow$ Global impact $\rightarrow$ Adapter re-generation

Con3. ip $\notin$ $\mathbb{B}$ $\wedge$ ip $\in$$(Out^A)_P$: A_{P, Q} can generate ip at t_A $\rightarrow$ AdaptSolution_1b (t_A)

AdaptSolution_1a (t_{AZ}) {

1 **if** ip $\notin$ $(Out^A)_P\prime$ **then** $(Out^A)_P\prime$:= $(Out^A)_P\prime$ $\cup$ {ip};

2 **if** t $\in$$UT_I^{P\prime}$ $\wedge$ $\exists$$t_{AX}$ ($\in$A_CrT) $\leftrightarrow$ t **then**

3 **if** ip $\in$ $\mathbb{B}$ (t_{AX}) **then**

4 $\mathbb{B}$:= $\mathbb{B}$ \ ip;

5 F_A := F_A $\cup$ $\{(t_{AX}, ip)_P\prime\}$;

6 **else** { Ident&Ext ($t_{AX,}$ t_{AZ}); F^A := F^A $\cup$$\{(t_{AN}, ip)_P\prime\}$;

7 **else if** t $\in$ $AT_I^{P\prime}$ $\wedge$ $\nexists$$t_{AX}$ ($\in$A_CrT) $\leftrightarrow$ t **then**

8 ExtendAd (t_{AZ}) ;

9 F^A := F^A $\cup$ $\{(t_{AN}, ip)_P\prime\}$;

}

AdaptSolution_1b (t_A) **{**

1 ExtendAd (t_A);

2 F^A := F^A $\cup$ $\{(t_{AN}, ip)_P\prime\}$;

}

ExtendAd (t_A) {

1 F^A := F^A \ $\{(t_A, t_A^\bullet)\}$;

2 P^A := P^A $\cup$$\{p_{AN}\}$;

3 T_I^A := T_I^A $\cup$ $\{t_{AN}\}$;

4 F^A := F^A $\cup$ $\{(t_A, p_{AN}), (p_{AN}, t_{AN}),(t_{AN}, t_A^\bullet)\}$;

5 **return** t_{AN}; **}**

Ident&Ext ($t_{AX,}$ t_{AZ}) {

1 **if** $\exists$ t_{AY} $\in$$T^A$ $\ni$ (# t_{AY} < # t_{AX} $\wedge$ ip $\in$ $\mathbb{B}$ (t_{AY})) **then**

2 ExtendAd (t_{AY});

3 **else** ExtendAd (t_{AZ}) ;

}

AAP #2 – Output Addition Pattern.

- *Evolution pattern*: *output interface place is added.* This pattern deals with the case where the output interface place *op* is newly added to the interface transition *t* of *P′*. For example, let the output message OrdConf of the transition SendOrderConfirmation is replaced by the output message OrdConf+Alternatives. Similar to AAP #1, the transition *t* is either a new interface transition receiving the output place *op*, or *t* is an existing interface transition in which the output (or input) interface place is replaced by *op*.
- *Potential impact*. The impact of this type of change of *P* on *A_{Q, P}* is partial.
- *Adaptation solution.* Adapter *A_{Q, P}* is extended on-the-fly by receiving *op* at the transition t_{AX} corresponding to *t* (i.e., $\exists\ t_{AX} \in A\_CrT \leftrightarrow t$). Also, the message *op* is stored in the $\mathbb{B}$ at the level of t_{AX}. For a new interface transition *t* where there is no transition t_{AX} of adapter corresponding to it, we propose a solution to identify an appropriate transition of *A_{Q, P}* to properly re-configure the adapter on-the-fly.

AAP #3 – Input Removed Pattern.

- *Evolution pattern*: *input interface place is removed.* In this pattern, the input interface place *ip* of the interface transition *t* in *P′* is removed. For example, suppose the input message Cancelation of the transition ReceiveCancelation is removed.
- *Potential impact*. The impact of this pattern on *A_{Q, P}* is partial.
- *Adaptation solution.* Adapter *A_{Q, P}* is dynamically updated to stop sending the respective message *ip* to *t* at the corresponding transition t_{AX}. Besides, the message *ip* is stored in the $\mathbb{B}$ at the level of t_{AX}. In the case where the transition *t* is also removed, we apply additional operations on *A_{Q, P}* to remove the corresponding transition t_{AX} and respective constituents from adapter (i.e., RedAdp function of Table 4.3).

AAP #4 – Output Removed Pattern.

- *Evolution pattern*: *output interface place is removed.* In this pattern, the output place *op* of the interface transition *t* of *P′* is removed. For example, the output message ShipNotif of the transition CreateShipmentNotification is removed.
- *Potential impact*. The impact of this type of evolution in *P′* on *A_{Q, P}* is partial.
- *Adaptation solution.* Adapter *A_{Q, P}* can be modified on-the-fly by stopping the reception of *op* at the transition t_{AX} corresponding to *t*, and updating the $\mathbb{B}$. Similar to AAP #3, if transition *t* is also removed, we execute additional operations on adapter.

Table 4.2. The specification of AAP #2.

Evolution Pattern: $(\exists t \in T_I^{P\prime} \wedge \exists op \in Out^{P\prime}) \ni op \in t^{\bullet}$ is a new output place
Potential Impact: Partial impact $\rightarrow$ AdaptSolution_2
AdaptSolution_2 { 1 **if** $op \notin (In^A)_{P\prime}$ **then** $(In^A)_{P\prime} := (In^A)_{P\prime} \cup \{op\}$; 2 **if** $t \in UT_I^{P\prime} \ni \exists t_{AX} (\in A\_CrT) \leftrightarrow t$ **then** 3 $F^A := F^A \cup \{(op, t_{AX})_{P\prime}\}$; 4 **if** $p \notin \mathbb{B}$ **then** 5 $\mathbb{B}(t_{AX}) := \mathbb{B}(t_{AX}) \cup \{op\}$; 6 **else if** $t \in AT_I^{P\prime} \ni \nexists t_{AX} (\in A\_CrT) \leftrightarrow t$ **then** 7 Ident&Ext_2(t_{AX}); 8 $F^A := F^A \cup \{(op, t_{AN},)_{P\prime}\}$; **}**
Ident&Ext_2 (t_n){ 1 **if** $\exists t' \in T_I^{P\prime} \ni (\#t' < \#t_n \wedge \exists t_{AY} (\in A\_CrT) \leftrightarrow t')$ **then** 2 ExtendAd (t_{AY}); 3 **else if** $\exists t' \in T_I^{P\prime} \ni (\#t' > \#t_n \wedge \exists t_{AZ} (\in A\_CrT) \leftrightarrow t')$ **then** 4 ExtendAd (t_{AZ}); **}**
ExtendAd (t_A) { 1 $F^A := F^A \setminus \{(t_A, t_A^{\bullet})\}$; 2 $P^A := P^A \cup \{p_{AN}\}$; 3 $T_I^A := T_I^A \cup \{t_{AN}\}$; 4 $F^A := F^A \cup \{(t_A, p_{AN}), (p_{AN}, t_{AN}), (t_{AN}, t_A^{\bullet})\}$; 4 **return** t_{AN}; **}**

Table 4.3. The specification of AAP #3.

<table>
<tr><td colspan="2">Evolution Pattern: $(\exists t \in T_I^{P\prime} \wedge \exists ip \in In^P) \ni$ (ip is a removed input place $\wedge$ $ip \notin {}^\bullet t$)</td></tr>
<tr><td colspan="2">Potential Impact: Partial impact $\rightarrow$ AdaptSolution_3 (t_{AX}) where $t_{AX} \in A\_CrT \leftrightarrow t$</td></tr>
<tr><td>AdaptSolution_3 (t_{AX}) {
1 $(Out^A)_{P\prime} := (Out^A)_{P\prime} \setminus \{ip\}$;
2 $F^A := F^A \setminus \{(t_{AX}, ip)_{P\prime}\}$;
3 if $p \notin \mathbb{B}$ then
4 $\mathbb{B}(t_{AX}) := \mathbb{B}(t_{AX}) \cup \{op\}$;
5 if $t \notin T_I^{P\prime}$ then
5 RedAdp(t_{AX});
}</td><td>RedAdp(t_A) {
1 $T_I^A := T_I^A \setminus \{t_A\}$;
2 if $\{t_A\} \subset \{({}^\bullet t_A)^\bullet\}$ then
3 $F^A := F^A \setminus \{({}^\bullet t_A, t_A), (t_A, t_A{}^\bullet)\}$;
4 else {
5 $P^A := P^A \setminus \{{}^\bullet t_A\}$;
6 $F^A := F^A \setminus \{({}^{\bullet\bullet} t_A, {}^\bullet t_A), ({}^\bullet t_A, t_A), (t_A, t_A{}^\bullet)\}$;
7 $F^A := F^A \cup \{({}^{\bullet\bullet} t_A, t_A{}^\bullet)\}$;
8 }
}</td></tr>
</table>

When an interface transition *t* is newly added to *P′*, accordingly *t* includes necessarily either an input or an output interface place. Therefore, the impact of this type of evolution on *P′* is evaluated through the AAPs #1 or #2, respectively. Moreover, in the case of the new interface transition, since no corresponding transition exists at adapter level, we propose a solution to identify the relevant transition of adapter for each of AAPs #1 or #2, separately. Accordingly, there is no need to take this type of change into consideration as a separate AAP. Furthermore, when an interface transition *t* is removed from *P′*, subsequently its input or its output place will be removed. Hence, the impact of this change is also dealt with the AAPs #3 or #4, respectively.

We provide the description of AAPs #1, #2, #3, and #4 in Tables 4.1, 4.2, 4.3, and 4.4 respectively. Obviously, when the changes of business protocols involve the respective evolution patterns of AAPs #2, #3, and #4, their potential impact on the current adapter would be partial. We first need a method to identify the relevant AAPs for business protocols evolution. Indeed, we detect the AAPs on a business protocol based on their respective evolution patterns. Hence, in the next section we describe our proposed method to detect the changes that can be occurred in a business protocol in terms of the evolution patterns.

Table 4.4. The specification of AAP #4.

<table>
<tr><td colspan="2">Evolution Pattern: $(\exists t \in T_I^{P\prime} \wedge \exists op \in Out^P) \ni$ (op is a removed output place $\wedge$ $op \notin t^\bullet$)</td></tr>
<tr><td colspan="2">Potential Impact: Partial impact $\rightarrow$ AdaptSolution_4 (t_{AX}) where $t_{AX} \in A\_CrT \leftrightarrow t$</td></tr>
<tr><td>AdaptSolution_4 {
1 $(In^A)_{P\prime} := (In^A)_{P\prime} \setminus \{op\}$;
2 $F^A := F^A \setminus \{(op, t_{AX})_{P\prime}\}$;
3 if $op \notin \mathbb{B}$ then
4 $\mathbb{B}(t_{AX}) := \mathbb{B}(t_{AX}) \cup \{op\}$;
6 if $t \notin T_I^{P\prime}$ then
7 RedAdp(t_{AX});
}</td><td>RedAdp(t_A) {
1 $T_I^A := T_I^A \setminus \{t_A\}$;
2 if $\{t_A\} \subset \{(^\bullet t_A)^\bullet\}$ then
3 $F^A := F^A \setminus \{(^\bullet t_A, t_A), (t_A, t_A^\bullet)\}$;
4 else {
5 $P^A := P^A \setminus \{^\bullet t_A\}$;
6 $F^A := F^A \setminus \{(^{\bullet\bullet} t_A, {}^\bullet t_A), (^\bullet t_A, t_A), (t_A, t_A^\bullet)\}$;
7 $F^A := F^A \cup \{(^{\bullet\bullet} t_A, t_A^\bullet)\}$;
8 }
}</td></tr>
</table>

4.2.2.2 Adapter adaptation patterns identification

In this section, we introduce a method to identify the AAPs on a business protocol evolution. We recognize the AAPs based on their respective evolution patterns. To detect the changes of a business protocol in terms of the evolution patterns, we apply a method on the basis of differences between the incidence matrices of the old and new version of respective oWFN, i.e., IM^P and $IM^{P'}$.

From this comparison, we first identify which interface transitions are added, removed or updated. For each one of these transitions, using boolean variable (*flag*), we then detect the types of changes in terms of added/removed input or output interface places (i.e., *f_AIn*, *f_AOut*, *f_RIn*, *f_ROut*, respectively). Therefore, with respect to the $IM^{P'}$, we acquire the set of added, removed, and updated interface transitions (i.e., AT_I, RT_I, and UT_I, respectively) and the types of changes on associated input/output interface places. The time complexity of these operations in the worst case can be expressed as $O(m^2)$ where $m = \max(|P_I|, |T_I|)$. In our approach, updated interface transitions represent the interface transitions which are also exist in the new version such that the evolution only occurs at the level of their respective interface places. In addition, the removed interface transition *t* indicates the interface transition which is no longer available in the new version, i.e., either it becomes an internal transition ($t \notin T_I^{P'}$), or it is completely removed from the list of transitions of P' ($t \notin T_I^{P'} \cap T^{P'}$). We denote by R_CT the set of interface transitions which are completely removed from P'.

Prior to the comparison of the matrices, we have to rearrange the respective rows and columns in both of them. Such a re-arrangement makes the columns and rows in both matrices are at the same order.

The method first compares the matrices IM^P and $IM^{P'}$ from point of view of their columns. Therefore, the method identifies the removed columns (removed interface transitions) if there is a column t_i (i.e., $0 < i < |T_I|$) in the IM^P such that any column t_i' equivalent to t_i does not exist in the $IM^{P'}$. The non-zero value of matrix IM^P at column t_i and row p_j (i.e., $0 < j < |P_I|$) signifies the type of changes in terms of removed input or removed output associated with this transition.

Similarly, the new interface transitions can be detected if a column t_i' exists in the $IM^{S'}$ such that IM^S does not include any equivalent column t_i to it. Accordingly, the non-zero value of matrix $IM^{S'}$ at row p_j' and column t_i' demonstrates the added input or output interface place to this transition.

Each pair of equivalent columns t_i (of IM^P) and t_i' (of $IM^{P'}$) is compared from point of view of the respective rows to identify the differences on the input or output interface places related to these interface transitions. Thus, for each two equivalent row p_j (of IM^P) and p_j' (of $IM^{P'}$) we compare the value of IM^P (at row p_j and columns t_i) with the value of $IM^{S'}$ (at row p_j' and columns t_i'). According to the value of matrices at corresponding row and column as depicted in Table 4.5, we detect the type of changes on associated input or output interface places. In such a way, we identify the updated interface transitions and respective changes on their input or output interface places.

Finally, we acquire the sets of changed interface transitions including added, removed, and updated interface transitions of P' (denoted by $CT_I^{P'}$), and also the type of changes on respective input/output interface places.

In the next section, we present an algorithm for evaluating the impact of business protocols evolution based on the change exploration, and the AAPs. The algorithm also dynamically updates the current adapter in the case of partial impact of an AAP.

Table 4.5. The flags with true value based on the value of Matrices IM^S and $IM^{S'}$

Value of IM ^{S} **at row *i* and column *j***	***Value of IM*** $^{S'}$ **at row *i'* and column *j'***	***True flags***
-1	0	*f_RIn*
0	-1	*f_AIn*
1	0	*f_ROut*
0	1	*f_AOut*

4.2.3 Protocol evolution impact analysis on the adapter

In this section, we present an algorithm that uses the specification of AAPs to automatically apply an impact analysis over the affected areas of new business protocol P'. To this end, the algorithm performs such an analysis through exploration and evaluation of the new, updated, and removed interface transitions in P' (denoted by *dirty* transitions) except for the removed interface transitions $t \in R_C T^{P'}$.

The algorithm starts from the initial state in P' and explores the dirty transitions according to the prefix order obtained by breadth-first traversal of the graph associated with the oWFN model. Once the algorithm meets a dirty transition, analyzes it to recognize its potential impact on the adapter. To do such an impact analysis, the algorithm uses the values of flags attributed to each dirty transition for identifying the respective evolution patterns of the AAPs. Therefore, for each dirty transition, the algorithm checks which flags are true, and which evolution patterns are then involved. According to the selected AAPs, specified impact analyses are performed.

The algorithm also performs dynamic adaptation solutions of the AAPs where the impact of relevant evolution patterns on the adapter is partial. Otherwise, if the impact is global, the algorithm stops the exploration of dirty transitions of P' and then results in the fact that the adapter should be re-generated from scratch. For each of the updated interface transitions, the algorithm first evaluates the impact of the removed input/output place, and then the impact of an added input/output place. In the case where an interface transition t is completely removed from P', the algorithm also executes additional tasks to accomplish the re-configuration of adapter, if dynamic updating the adapter is possible.

Algorithm 2 shows the proposed method of evolution impact analysis for AAP #1. We have left out the bfs-based exploration of dirty transitions in P'.

Algorithm 2 takes as input the oWFN model of protocols P, P', and adapter $A\_\{Q, P\}$, and also the incidence matrices IM^P and $IM^{P'}$. The set $dT^{P'}$ stands for dirty transitions of P' (i.e., $dT^{P'} := UT_I{}^{P'} \cup AT_I{}^{P'} \cup RT_I{}^{P'} \setminus R_C T^{P'}$). The set T_C initiates by the union of the sets $dT^{P'}$ and $R_C T^{P'}$. For each t of $dT^{P'}$, in the case of AAP #1, algorithm 2 first verifies whether flag *f_AIn* (standing for its evolution pattern) related to t is true. Then, algorithm 2 detects the input interface place p of P' added to t using the $IM^{P'}$ where the element at row p and column t is the value "-1" (line 2). According to the specification of $A\_\{Q, P\}$, the algorithm analyses the potential impact of AAP #1 (line 3).

The algorithm then applies either the function adaptSolution_1a or adaptSolution_1b to dynamically update $A\_\{Q, P\}$, if the impact of pattern is partial. Otherwise, if the impact is global, the algorithm recognizes that $A\_\{Q, P\}$ should be re-developed. Thus, the exploration of P' (line 5) is stopped. For every interface transition t of P' which is completely removed (i.e., $t \in R_C T^{P'}$ and cannot be explored), the algorithm finally performs function extraAdapt to complete the re-configuration of $A\_\{Q, P\}$ (lines 7-8). Given the type of interface place on the removed transition t, the algorithm executes appropriate operations

to update the adapter properly. Algorithm 2 finally either yields to a new adapter $A\_\{Q, P'\}$ on-the-fly or realizes that adapter $A\_\{Q, P\}$ should be re-generated at design-time.

Algorithm 2: Impact analysis from point of view of P′ on adapter A_{Q, P}

```
1   dT^P' = CT_I^P' \ R_C T^P' ;
2   while dT^P' ≠ ∅ ∋ ∃t ∈ dT^P' do
3       if t_f_AIn = true ∋ ∃p ∈ In^P' with IM^P' [p, t] < 0 then // AAP #1
4           ImpactAnalysis;        // Potential Impact of Table 1
5           if impact is global then
6               break;
7       dT^P' := dT^P' \ t ;
8   if impact is partial ∧ ( T_C \ dT^P' ≠ ∅ ∋ ∃ t ∈R_C T^P') then
8     extraAdapt(t);

9     extraAdapt (t) {
10        if ∃p ∈ Out^P with IM^P[p, t] > 0 then
11            AdaptSolution_4 (line 1-4) ;
12            RedAd(t_A) ;
13        else if ∃p ∈ In^P with IM^P[p, t] < 0 then
14            AdaptSolution_3 (line 1-4) ;
15            RedAd(t_A) ;
16  }
```

Use-Case. We suppose the evolution of protocol P occurs in two use-cases α and β of P'.

- The changes of P' in the case α comprise of:
  1) The new interface transition AnalyzeBuyerHistory (i.e., which is available following the interface transition ProcessOrder) receives the input message BuyerInfo; and
  2) The input message BuyerInfo of interface transition ProcessByuerInformation is removed in which this transition is also eliminated (i.e., ProcessBuyerInformation $\in R_C T_I^{P'}$).
- In the case β protocol P' receives the input message InvoiceConf by the new interface transition ReceiveInvoiceConfirmation (i.e., prior to the transition ReceivePayment). In the following, we apply Algorithm 1 on both cases α and β of P'.

In the case α of P', Algorithm 2 starts from the initial state p_0 and then explores the dirty transition AnalyzeBuyerHistory which is a new interface transition (i.e., AnalyzeBuyerHistory $\in AT_I^{P'}$) receiving the input message BuyerInfo (i.e., flag *f_AIn* of AnalyzeBuyerHistory is true). Therefore, the algorithm evaluates the potential impact of AAP #1 for this transition as follows:

- The buffer of $A\_\{Q, P\}$ covers BuyerInfo at the transition tA_6. Accordingly, the algorithm calls function adaptSolution_1a to extend $A\_\{Q, P\}$ at tA_6. This function creates the new transition t_{AN} and flow relation $(t_{AN}, \text{BuyerInfo})_{P'}$ which sends BuyerInfo to the transition AnalyzeBuyerHistory). In addition, the buffer of $A\_\{Q, P\}$ is updated.
- For the dirty removed interface transition ProcessBuyerInformation that is also unobservable in P' (i.e., ProcessBuyerInformation $\in R_C T^{P'}$), the algorithm calls the function extraAdapt on it. This function performs final adjustments on $A\_\{Q, P\}$ at the corresponding transition tA_8 to stop sending the respective message BuyerInfo to P' (i.e., flag *f_RIn* of ProcessBuyerInformation is true). This function also applies the function RedAd on tA_8 of $A\_\{Q, P\}$ to remove this transition and its respective elements from adapter. Consequently, the new adapter $A\_\{Q, P'\}$ is dynamically available.

In the case β of P', algorithm 2 visits and analyzes the dirty new interface transition ReceiveInvoiceConfirmation receiving the input message InvoiceConf based on AAP #1. Respecting the specification of $A\_\{Q, P\}$, buffer $\mathbb{B}$ includes InvoiceConf at transition tA_{12}. The impact of pattern is partial. Therefore, the algorithm applies function adaptSolution_1a on $A\_\{Q, P\}$ where extends adapter at transition tA_{12} to send this message to transition ReceiveInvoiceConfirmation of P'. Similar to the case α, new adapter $A\_\{Q, P'\}$ is provided on-the-fly.

In the next section, we propose a method to verify the correctness of adapter $A\_\{Q, P'\}$ that is dynamically formed in both cases α and β. In our approach, adapter $A\_\{Q, P'\}$ is correct with respect to P'

and Q if it is behaviorally compatible with both of them (i.e., incorrect adapter here means a protocol mismatch in terms of deadlock exists between the adapter and one of the participant partners).

4.2.4 Dynamic verification of adapted adapter

This section is devoted to verify the correctness of the new adapter which is dynamically provided. Adapter verification is needed in most adapter re-generation scenarios, except those which guarantee the adapter soundness, e.g., as in Yellin and Strom [20]. In our approach, the verification phase is performed on-the-fly once the adaptation of current adapter is done. The proposed verification is based on structural reasoning related to necessary condition of deadlock occurrence which is equivalent to the existence of *unmarked siphon* [46] in the corresponding oWFN $\langle Q \oplus A\_\{Q, P'\} \oplus P' \rangle$ (i.e., modeling the interaction among the new adapter $A\_\{Q, P'\}$, and partners Q and P'). We take the advantages of this structural verification to avoid the state-space checking which is computationally expensive in general. This structural verification can be done in polynomial time that consists in removing the initially marked places, and checking the existence or non-existence of an unmarked siphon in the resulting subnet.

We recall that a nonempty place set $S \subseteq P$ of a P/T net N is called a siphon if and only if ${}^\bullet S \subseteq S^\bullet$. S is said to be minimal if and only if it contains no other siphon as a proper subset. N is said to be satisfying the controlled-siphon property (CS-property) if and only if all its minimal siphons are controlled. Therefore, we have the following results:

- In the case where there is no an unmarked siphon, and also if the corresponding oWFN belongs to the class of K-systems such as root nets or ordered nets [28], we can claim that protocol compatibility is guaranteed. Indeed, for such classes of nets the CS-property is not only necessary liveness condition but also sufficient, and for general nets the necessary compatibility condition is satisfied (i.e., the adapter is quasi-correct).
- In the case of existence of siphon, we identify the incompatibility problem with respect to the siphon places, and this information is valid for general cases (i.e., any nets). Besides, this information can be fully exploited for correction bases (e.g. adaptation solutions substitution, or in adapter re-generation method if such a method is inevitable).

For instance, in the case α of P' (as described in the previous section) there is an unmarked siphon on the corresponding oWFN $\langle Q \oplus A\_\{Q, P'\} \oplus P' \rangle$ where involves places BuyerInfo and OrdConf (i.e., P' receives BuyerInfo before sending OrdConf). Hence, the CS-property is not satisfied and final marking $M_f = q_7 + a_{14} + p_6$ is unreachable; while in the case β of P', the correctness of $A\_\{Q, P'\}$ is recognized.

4.2.5 Prototype implementation and experiments

4.2.5.1 Implementation

The presented approach has been implemented as a prototype tool in the context of PIPE2 (Platform Independent Petri net Editor 2) architecture, which is an open source, and Java-based tool for creating and analyzing Petri nets efficiently [29].

Figure 4.4 represents the architecture of the prototype. We first applied the free tool BPEL2oWFN [50] to automatically translate the partner business protocols into oWFNs in the format of PNML (Petri Net Markup Language) [148]. We then made some modifications on the specifications of BPEL2oWFN's output files (XML files) to meet the PIPE2's inputs requirements. Since PIPE2 conforms fully to PNML (i.e., the file format is extensible through the use of XSLT), we extended the model of place and transition at various parts of PIPE2's sources to support the interface transitions and places of an oWFN.

The execution of our module – *Evolution-aware Impact Analysis* begins by reception of the original versions of the partner business protocols (e.g., *P.XML* and *Q.XML*), and also the new version of *P* (e.g., *P′.XML*). With ex-ante knowledge of incompatible interaction between *Q* and *P*, the adapter *A.XML* is called (i.e., *compatibility checking* component will be added in the future). The same method has been applied to produce an appropriate version of *A.XML* to be used in the PIPE2. We have also extended the incidence matrices of PIPE2 to allow to create the incidence matrices IM^{P} and $IM^{P'}$. We compare these matrices by *comparison* component to identify the differences between these versions in the form of AAPs. Finally, the *impact analysis* component provides the results of evaluating the impact of P' on the adapter based on both *A.XML* and the list of selected AAPs.

Our tool assists the adapter developers to decide efficiently how to deal with the adapter in the context of business protocols evolution.

4.2.5.2 Experimental evaluation

In support of the proposed approach, we have applied the prototype tool on some synthetic scenarios. In our experiments, we have taken the specification of all AAPs into consideration. Our investigation shows that the proposed approach is advantageous to make crucial decisions on when we can prevent from complete adapter regeneration where dynamic updating the adapter meets the impact of evolution.

To realize the effectiveness of proposed approach, we have made a comparison between our proposed approach and the static adapter regeneration method [149]. To this end, we have estimated the time complexity of both of these approaches. The results of this comparison demonstrate the circumstances where if the changes of business protocols have partial impact on current adapter, the proposed approach is more effective in cost saving.

The time complexity for a method of complete adapter regeneration is O(m^2) where m is the number of interface transitions. The proposed evolution-aware impact analysis for each of the AAPs has the time complexity of O(m). Moreover, the time complexity of the proposed solutions of dynamic adaptation is O(m). In the case where a change with global impact exists in a business protocol, both complete adaptation and our dynamic approach have similar time complexity O(m^2).

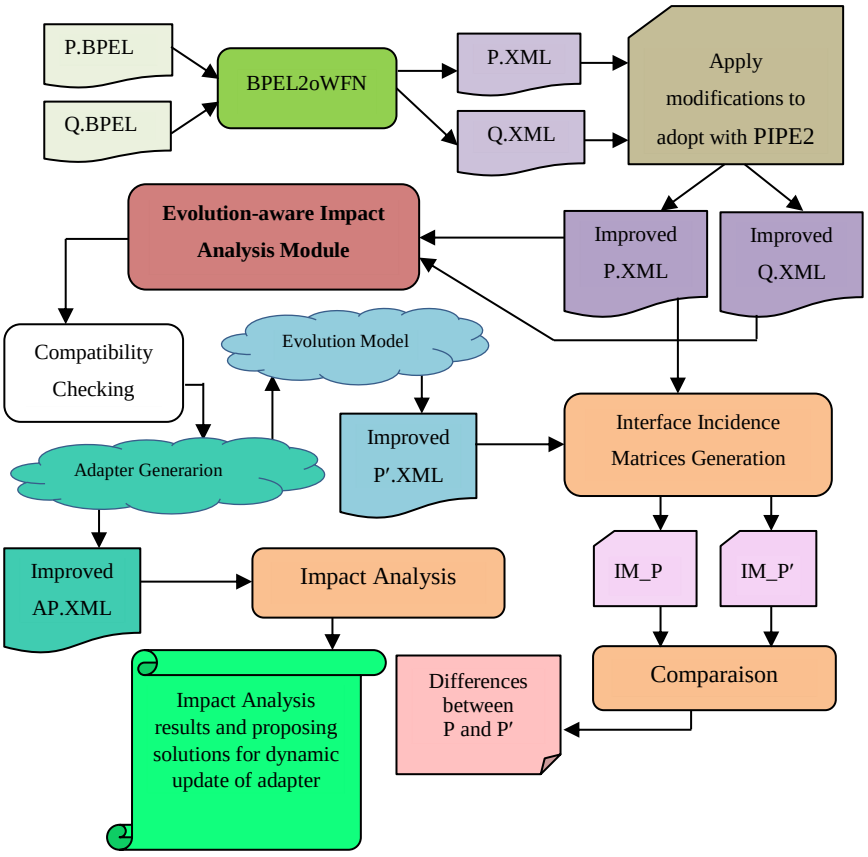

Figure 4.4. The architecture of prototype implementation

Conclusions

In this chapter, we discuss a summary of the contributions. We then conclude the thesis by providing directions for future extensions of the presented approach.

1 Summery of contributions

The first part of this thesis has been dedicated to the verification of compatibility of composite services. In our proposed approach, we structurally verified the compatibility of service composition using the concepts and structure theory of Petri Nets. To this end, the verification method was based on analyzing the existence of unmarked siphon in the corresponding composit net. In addition, we then discussed the needs for adaptation of collaboratable services to make their incompatible interactions possible. We also illustrated an overview of service adaptation approaches which are dealing with mismatches at the level of their interface and behavior.

In the second part, we concentrated on the main contribution of this thesis relating to the evolution of partner services for those which their interactions are made by an adapter module. We first discussed the related works to the context of evolution of services involved in an interoperability. We then investigated the challenge of adapters' adaptation to comply with the evolution of participant services particulaly when their business protocols evolve. On this concern, we introduced the common adapter adaptation patterns of the evolution of service business protocols. We presented a method to automatically identify these patterns in a business protocol participating in an adapter. We mainly established an algorithm to evaluate the impact of such an evolution of partner services on the adapter. When this analyser algorithm recognizes the fact that the impact of evolution on the adapter is not global, the algorithm allows the re-configuration of current adapter on-the-fly based on the adaptation patterns.

Furthermore, we applied a structural technique to verify the correctness of the new adapter which is dynamically updated. If the verification technique recognizes that the protocol compatibility between the new adapter and respective partners cannot be preserved, the developers then can take the advantage of this structural information to identify the source of mismatches. Hence, with having such an evolution-aware analysis method, they can efficiently make an appropriate adapter at design-time compared to adapter re-

generation methods. Another result of this work is to identify the type of business protocol evolution enabling the adapter developers to avoid the complete adapter re-generation.

We finally presented a prototype tool implementing the proposed approach. The obtained experimental results show the effectiveness of our approach.

2 Future directions

This section evaluates the extendability of the core approach presented in this thesis.

As we pointed out already, our approach has been proposed based upon the type of oWFNs in which each interface transition has only one input or one output interface place. In addition, no directed cycle exists in the oWFN modeling of the adapter and service business protocols (before and after evolution). Therefore, as future work, the proposed algorithm of evolution detection in the oWFN model of business protocols can be optimized by relaxation of acyclic constraint.

In our contribution, we assumed that only one of the partner services' business protocols evolves. Therefore, one may desire to take the evolution of both partners' business protocols into consideration. Moreover, the evolution of service business protocol can be seen from a different standpoint. Accordingly, the proposed adapter adaptation patterns (AAPs) could be changed. These changes may also influence dynamic re-configuration of the adapter.

In addition, one may focus on how the security policy can be preserved by composition of linear oWFNs. Besides, the proposed compatibility verification approach can be extended under time-constraints (i.e., timed oWFNs).

The scalability of proposed approach can also be improved by applying more real-world scenarios.

Bibliography

[1] M. P. Papazoglou, “Agent-oriented technology in support of e-business,” *Commun. ACM*, vol. 44, no. 4, pp. 71–77, 2001.

[2] M. P. Papazoglou, “Service-oriented computing: Concepts, characteristics and directions,” in *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, 2003, pp. 3–12.

[3] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services: Concepts, Architectures and Applications*. Springer Publishing Company, Incorporated, 2010.

[4] D. Austin, A. Barbir, C. Ferris, and S. Garg, “Web services architecture requirements,” *W3C Work. Draft*, vol. 19, 2002.

[5] B. Medjahed, A. Rezgui, A. Bouguettaya, and M. Ouzzani, “Infrastructure for e-government web services,” *Internet Comput. IEEE*, vol. 7, no. 1, pp. 58–65, 2003.

[6] D. Jordan, J. Evdemon, A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, and Y. Goland, “Web services business process execution language version 2.0,” *OASIS Stand.*, vol. 11, 2007.

[7] K. Gottschalk, “Web Services architecture overview,” *IBM Whitepaper IBM Dev.*, vol. 1, 2000.

[8] T. Erl, *Service-oriented architecture*, vol. 8. Prentice Hall New York, 2005.

[9] R. J. Glushko and T. McGrath, “Document Engineering: analyzing and designing the semantics of Business Service Networks,” in *Proceedings of the IEEE EEE05 international workshop on Business services networks*, 2005, pp. 2–2.

[10] M. P. Papazoglou and W.-J. Van Den Heuvel, “Service oriented architectures: approaches, technologies and research issues,” *VLDB J.*, vol. 16, no. 3, pp. 389–415, 2007.

[11] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, *Simple object access protocol (SOAP) 1.1*. 2000.

[12] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana, “Web services description language (wsdl) version 2.0 part 1: Core language,” *W3C Recomm.*, vol. 26, 2007.

[13] T. Bellwood, S. Capell, and J. Colgrave, “Universal Description, Discovery and Integration specification (UDDI) 3.0,” *Online Httpuddi Orgpubsuddi-V3 00-Publ.-20020719 Htm*, 2002.

[14] P. Felber, C.-Y. Chan, M. Garofalakis, and R. Rastogi, “Scalable filtering of XML data for Web services,” *Internet Comput. IEEE*, vol. 7, no. 1, pp. 49–57, 2003.

[15] B. Hofreiter, C. Huemer, and W. Klas, “ebXML: status, research issues, and obstacles,” in *Research Issues in Data Engineering: Engineering E-Commerce/E-Business Systems, 2002. RIDE-2EC 2002. Proceedings. Twelfth International Workshop on*, 2002, pp. 7–16.

[16] S. Dustdar and W. Schreiner, “A survey on web services composition,” *Int. J. Web Grid Serv.*, vol. 1, no. 1, pp. 1–30, 2005.

[17] J. Rao and X. Su, “A survey of automated web service composition methods,” in *Semantic Web Services and Web Process Composition*, Springer, 2005, pp. 43–54.

[18] H. R. M. Nezhad, B. Benatallah, F. Casati, and F. Toumani, “Web services interoperability specifications,” *IEEE Comput. Soc.*, vol. 39, no. 5, pp. 24–32, 2006.

[19] L. Bordeaux, G. Sala\ün, D. Berardi, and M. Mecella, “When are two web services compatible?,” *Technol. E-Serv.*, pp. 15–28, 2005.

[20] D. M. Yellin and R. E. Strom, "Protocol specifications and component adaptors," *ACM Trans. Program. Lang. Syst. TOPLAS*, vol. 19, no. 2, pp. 292–333, 1997.

[21] Z. Zhou, S. Bhiri, W. Gaaloul, and M. Hauswirth, "Developing process mediator for supporting mediated web service interactions," in *IEEE Sixth European Conference on Web Services ECOWS'08*, 2008, pp. 155–164.

[22] H. R. Motahari Nezhad, G. Y. Xu, and B. Benatallah, "Protocol-aware matching of web service interfaces for adapter development," in *Proceedings of the 19th international conference on World Wide Web*, 2010, pp. 731–740.

[23] B. Benatallah, F. Casati, D. Grigori, H. R. . Nezhad, and F. Toumani, "Developing adapters for web services integration," in *Advanced Information Systems Engineering*, 2005, pp. 415–429.

[24] Z. Feng, K. He, R. Peng, and Y. Ma, "Taxonomy for Evolution of Service-Based System," in *IEEE World Congress on Services (SERVICES)*, 2011, pp. 331–338.

[25] C. Courbis and A. Finkelstein, "Towards aspect weaving applications," in *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*, 2005, pp. 69–77.

[26] P. Massuthe, W. Reisig, and K. Wolf, *An operating guideline approach to the SOA*. Annals of Mathematics, Computing & Teleinformatics 1(3), 35–43, 2005.

[27] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[28] K. Barkaoui, R. B. Ayed, and Z. Sbaï, "Workflow soundness verification based on structure theory of Petri nets," *Int. J. Comput. Inf. Sci.*, vol. 5, no. 1, pp. 51–61, 2007.

[29] P. Bonet, C. M. Lladó, R. Puijaner, and W. J. Knottenbelt, "PIPE v2. 5: A Petri net tool for performance modelling," in *Proc. 23rd Latin American Conference on Informatics (CLEI 2007)*, 2007.

[30] M. Eslamichalandar, K. Barkaoui, and H. R. Motahari-Nezhad, "Service Composition Adaptation: an Overview."

[31] K. Barkaoui, M. Eslamichalandar, and M. Kaabachi, "A structural verification of web services composition compatibility," in *Proceedings of the 6th International Workshop on Enterprise & Organizational Modeling and Simulation*, 2010, pp. 30–41.

[32] D. Beyer, A. Chakrabarti, and T. A. Henzinger, "Web service interfaces," in *Proceedings of the 14th international conference on World Wide Web*, 2005, pp. 148–159.

[33] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Sci. Am.*, vol. 284, no. 5, pp. 28–37, 2001.

[34] F. Bonchi, A. Brogi, S. Corfini, and F. Gadducci, "Compositional specification of web services via behavioural equivalence of nets: A case study," in *Applications and Theory of Petri Nets*, Springer, 2008, pp. 52–71.

[35] C. Ouyang, E. Verbeek, W. M. Van Der Aalst, S. Breutel, M. Dumas, and A. H. Ter Hofstede, "Formal semantics and analysis of control flow in WS-BPEL," *Sci. Comput. Program.*, vol. 67, no. 2, pp. 162–198, 2007.

[36] N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg, "Analyzing interacting WS-BPEL processes using flexible model generation," *Data Knowl. Eng.*, vol. 64, no. 1, pp. 38–54, 2008.

[37] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[38] R. Hamadi and B. Benatallah, "A Petri net-based model for web service composition," in *Proceedings of the 14th Australasian database conference-Volume 17*, 2003, pp. 191–200.

[39] X. Fu, T. Bultan, and J. Su, "Analysis of interacting BPEL web services," in *Proceedings of the 13th international conference on World Wide Web*, 2004, pp. 621–630.

[40] B. Benatallah, F. Casati, and F. Toumani, "Representing, analysing and managing web service protocols," *Data Knowl. Eng.*, vol. 58, no. 3, pp. 327–357, 2006.

[41] A. Ferrara, "Web services: a process algebra approach," in *Proceedings of the 2nd international conference on Service oriented computing*, 2004, pp. 242–251.

[42] R. Mateescu, P. Poizat, and G. Salaün, "Behavioral adaptation of component compositions based on process algebra encodings," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, 2007, pp. 385–388.
[43] R. Milner, *The polyadic π-calculus: a tutorial*. Springer, 1993.
[44] C. A. Petri, "Kommunikation mit automaten," 1962.
[45] S. Hinz, K. Schmidt, and C. Stahl, "Transforming BPEL to Petri nets," *Bus. Process Manag.*, pp. 220–235, 2005.
[46] K. Barkaoui, J. M. Couvreur, and K. Klai, "On the equivalence between liveness and deadlock-freeness in Petri nets," *Appl. Theory Petri Nets 2005*, pp. 90–107, 2005.
[47] W. van der Aalst, K. van Hee, P. Massuthe, N. Sidorova, and J. van der Werf, "Compositional service trees," *Appl. Theory Petri Nets*, pp. 283–302, 2009.
[48] W. Reisig, J. Bretschneider, D. Fahland, N. Lohmann, P. Massuthe, and C. Stahl, "Services as a Paradigm of Computation," *Form. Methods Hybrid Real-Time Syst.*, pp. 521–538, 2007.
[49] W. van der Aalst, "Verification of workflow nets," *Appl. Theory Petri Nets 1997*, pp. 407–426, 1997.
[50] N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg, "Analyzing interacting BPEL processes," *Bus. Process Manag.*, pp. 17–32, 2006.
[51] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, R. Metz, and B. A. Hamilton, "Reference model for service oriented architecture 1.0," *OASIS Stand.*, vol. 12, 2006.
[52] B. Medjahed, B. Benatallah, A. Bouguettaya, A. H. Ngu, and A. K. Elmagarmid, "Business-to-business interactions: issues and enabling technologies," *VLDB Journal— Int. J. Very Large Data Bases*, vol. 12, no. 1, pp. 59–85, 2003.
[53] B. Benatallah, F. Casati, and F. Toumani, "Web service conversation modeling: A cornerstone for e-business automation," *Internet Comput. IEEE*, vol. 8, no. 1, pp. 46–54, 2004.
[54] C. Peltz, "Web services orchestration and choreography," *Computer*, vol. 36, no. 10, pp. 46–52, 2003.
[55] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto, "Web services choreography description language version 1.0," *W3C Candidate Recomm.*, vol. 9, 2005.
[56] G. Decker, O. Kopp, F. Leymann, and M. Weske, "BPEL4Chor: Extending BPEL for modeling choreographies," in *Web Services, 2007. ICWS 2007. IEEE International Conference on*, 2007, pp. 296–303.
[57] C. Yushi, L. E. Wah, and D. K. Limbu, "Web Services Composition-An Overview of Standards," *Singap. Inst. Manuf. Technol. Sect. Four*, p. 10, 2004.
[58] W. van der Aalst and M. Weske, "The P2P approach to interorganizational workflows," in *Advanced Information Systems Engineering*, 2001, pp. 140–156.
[59] J. Yu, T. Manh, J. Han, Y. Jin, Y. Han, and J. Wang, "Pattern based property specification and verification for service composition," *Web Inf. Syst. 2006*, pp. 156–168, 2006.
[60] K. Klai and D. Poitrenaud, "MC-SOG: An LTL model checker based on symbolic observation graphs," in *Applications and Theory of Petri Nets*, Springer, 2008, pp. 288–306.
[61] K. Klai, S. Tata, and J. Desel, "Symbolic abstraction and deadlock-freeness verification of inter-enterprise processes," *Data Knowl. Eng.*, 2011.
[62] E. A. Emerson, *Temporal and modal logic, Handbook of theoretical computer science (vol. B): formal models and semantics*. MIT Press, Cambridge, MA, 1991.
[63] B. Benatallah, M. Dumas, and Q. Z. Sheng, "Facilitating the rapid development and scalable orchestration of composite web services," *Distrib. Parallel Databases*, vol. 17, no. 1, pp. 5–37, 2005.
[64] M. P. Papazoglou, "Web services and business transactions," *World Wide Web*, vol. 6, no. 1, pp. 49–91, 2003.
[65] S. Dalal, S. Temel, M. Little, M. Potts, and J. Webber, "Coordinating business transactions on the web," *Internet Comput. IEEE*, vol. 7, no. 1, pp. 30–39, 2003.
[66] A. Alamri, M. Eid, and A. El Saddik, "Classification of the state-of-the-art dynamic web services composition techniques," *Int. J. Web Grid Serv.*, vol. 2, no. 2, pp. 148–166, 2006.

[67] E. Truyen, B. N. Jørgensen, W. Joosen, and P. Verbaeten, “On interaction refinement in middleware,” 2000.

[68] G. Spanoudakis, A. Zisman, and A. Kozlenkov, “A service discovery framework for service centric systems,” in *Services Computing, 2005 IEEE International Conference on*, 2005, vol. 1, pp. 251–259.

[69] H. Skogsrud, B. Benatallah, and F. Casati, “Trust-serv: model-driven lifecycle management of trust negotiation policies for web services,” in *Proceedings of the 13th international conference on World Wide Web*, 2004, pp. 53–62.

[70] R. Quintero, V. Torres, and V. Pelechano, “Model centric approach of web services composition,” in *Emerging Web Services Technology*, Springer, 2007, pp. 65–81.

[71] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M.-C. Shan, “Adaptive and dynamic service composition in eFlow,” in *Advanced Information Systems Engineering*, 2000, pp. 13–31.

[72] A. Barros, M. Dumas, and A. H. Ter Hofstede, “Service interaction patterns,” in *Business Process Management*, Springer, 2005, pp. 302–318.

[73] W. M. van der Aalst, A. J. Mooij, C. Stahl, and K. Wolf, “Service interaction: Patterns, formalization, and analysis,” in *Formal Methods for Web Services*, Springer, 2009, pp. 42–88.

[74] R. Hull and J. Su, “Tools for composite web services: a short overview,” *ACM SIGMOD Rec.*, vol. 34, no. 2, pp. 86–95, 2005.

[75] K. van Hee, N. Sidorova, and J. van der Werf, “Construction of asynchronous communicating systems: weak termination guaranteed!,” in *Software Composition*, 2010, pp. 106–121.

[76] D. Zhovtobryukh, “A petri net-based approach for automated goal-driven web service composition,” *Simulation*, vol. 83, no. 1, p. 33, 2007.

[77] W. M. . Aalst, N. Lohmann, M. La Rosa, and J. Xu, “Correctness ensuring process configuration: an approach based on partner synthesis (extended version),” 2010.

[78] O. Oanea and K. Wolf, “An efficient necessary condition for compatibility,” in *ZEUS*, vol. 438.

[79] X. Li, Y. Fan, Q. Z. Sheng, Z. Maamar, and H. Zhu, “A petri net approach to analyzing behavioral compatibility and similarity of web services,” *Syst. Man Cybern. Part Syst. Humans IEEE Trans.*, vol. 41, no. 3, pp. 510–521, 2011.

[80] Z. Wu, S. Deng, Y. Li, and J. Wu, “Computing compatibility in dynamic service composition,” *Knowl. Inf. Syst.*, vol. 19, no. 1, pp. 107–129, 2009.

[81] M. De Backer, “On the verification of Web services compatibility: a Petri Net approach,” in *On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops*, 2004, pp. 810–821.

[82] A. Martens, “On compatibility of web services,” *Petri Net Newsl.*, vol. 65, pp. 12–20, 2003.

[83] N. Lohmann, “Correcting deadlocking service choreographies using a simulation-based graph edit distance,” in *Business Process Management*, Springer, 2008, pp. 132–147.

[84] H. Foster, S. Uchitel, J. Magee, and J. Kramer, “Compatibility verification for web service choreography,” in *Web Services, 2004. Proceedings. IEEE International Conference on*, 2004, pp. 738–741.

[85] N. Lohmann, O. Kopp, F. Leymann, and W. Reisig, “Analyzing BPEL4Chor: Verification and participant synthesis,” in *Web Services and Formal Methods*, Springer, 2008, pp. 46–60.

[86] A. Martens, S. Moser, A. Gerhardt, and K. Funk, “Analyzing compatibility of bpel processes,” 2006.

[87] K. Wolf, “Does my service have partners?,” in *Transactions on Petri Nets and Other Models of Concurrency II*, Springer, 2009, pp. 152–171.

[88] D. Weinberg, “Efficient controllability analysis of open nets,” in *Web Services and Formal Methods*, Springer, 2009, pp. 224–239.

[89] N. Lohmann and D. Weinberg, “Wendy: A tool to synthesize partners for services,” in *Applications and theory of Petri nets*, Springer, 2010, pp. 297–307.

[90] P. C. Xiong, Y. S. Fan, and M. C. Zhou, “A Petri net approach to analysis and composition of web services,” *IEEE Trans. Syst. Man Cybern. Part Syst. Humans*, vol. 40, no. 2, pp. 376–387, 2010.

[91] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. The MIT press, 1999.

[92] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "Nusmv 2: An opensource tool for symbolic model checking," in *Computer Aided Verification*, 2002, pp. 359–364.

[93] T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre, "Software verification with BLAST," in *Model Checking Software*, Springer, 2003, pp. 235–239.

[94] G. J. Holzmann, "The model checker SPIN," *Softw. Eng. IEEE Trans.*, vol. 23, no. 5, pp. 279–295, 1997.

[95] H. Dun, H. Xu, and L. Wang, "Transformation of BPEL processes to Petri Nets," in *Theoretical Aspects of Software Engineering, 2008. TASE'08. 2nd IFIP/IEEE International Symposium on*, 2008, pp. 166–173.

[96] W. M. van der Aalst, "The application of Petri nets to workflow management," *J. Circuits Syst. Comput.*, vol. 8, no. 01, pp. 21–66, 1998.

[97] K. Barkaoui, J.-M. Couvreur, and K. Klai, "On the equivalence between liveness and deadlock-freeness in Petri nets," in *Applications and Theory of Petri Nets 2005*, Springer, 2005, pp. 90–107.

[98] W. Kongdenfha, R. Saint-Paul, B. Benatallah, and F. Casati, "An aspect-oriented framework for service adaptation," *Serv.-Oriented Comput. 2006*, pp. 15–26, 2006.

[99] R. Mateescu, P. Poizat, and G. Sala\ün, "Adaptation of service protocols using process algebra and on-the-fly reduction techniques," *Serv.-Oriented Comput. 2008*, pp. 84–99, 2008.

[100] A. J. Mooij and M. Voorhoeve, "Proof techniques for adapter generation," in *Web Services and Formal Methods*, Springer, 2009, pp. 207–223.

[101] W. Tan, Y. Fan, and M. C. Zhou, "A petri net-based method for compatibility analysis and composition of web services in business process execution language," *IEEE Trans. Autom. Sci. Eng.*, vol. 6, no. 1, pp. 94–106, 2009.

[102] J. A. Martín and E. Pimentel, "Automatic generation of adaptation contracts," *Electron. Notes Theor. Comput. Sci.*, vol. 229, no. 2, pp. 115–131, 2009.

[103] M. Ouederni, G. Salaün, and E. Pimentel, "Measuring the compatibility of service interaction protocols," in *Proceedings of the 2011 ACM Symposium on Applied Computing*, 2011, pp. 1560–1567.

[104] Z. Zhou, S. Bhiri, H. Zhuge, and M. Hauswirth, "Assessing service protocol adaptability based on protocol reduction and graph search," *Concurr. Comput. Pr. Exp.*, vol. 23, no. 9, pp. 880–904, 2011.

[105] L. Cavallaro, E. Di Nitto, P. Pelliccione, M. Pradella, and M. Tivoli, "Synthesizing adapters for conversational web-services from their WSDL interface," in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, 2010, pp. 104–113.

[106] C. Canal, P. Poizat, and G. Salaun, "Model-based adaptation of behavioral mismatching components," *Softw. Eng. IEEE Trans.*, vol. 34, no. 4, pp. 546–563, 2008.

[107] X. Li, Y. Fan, S. Madnick, and Q. Z. Sheng, "A pattern-based approach to protocol mediation for web services composition," *Inf. Softw. Technol.*, vol. 52, no. 3, pp. 304–323, 2010.

[108] W. Kongdenfha, H. R. Motahari-Nezhad, B. Benatallah, F. Casati, and R. Saint-Paul, "Mismatch patterns and adaptation aspects: A foundation for rapid development of web service adapters," *IEEE Trans. Serv. Comput.*, pp. 94–107, 2009.

[109] K. W. Wang, "Interface adaptation for conversational services," 2008.

[110] H. W. Schmidt and R. H. Reussner, "Generating adapters for concurrent component protocol synchronisation," in *Proceedings of the IFIP TC6/WG6*, 2002, vol. 1, pp. 213–229.

[111] A. Brogi and R. Popescu, "Automated generation of BPEL adapters," *Serv.-Oriented Comput. 2006*, pp. 27–39, 2006.

[112] N. Guermouche, O. Perrin, and C. Ringeissen, "A mediator based approach for services composition," in *Software Engineering Research, Management and Applications, 2008. SERA'08. Sixth International Conference on*, 2008, pp. 273–280.

[113] Q. Z. Sheng, B. Benatallah, M. Dumas, and E. O.-Y. Mak, "SELF-SERV: A platform for rapid composition of web services in a peer-to-peer environment," in *Proceedings of the 28th international conference on Very Large Data Bases*, 2002, pp. 1051–1054.

[114] L. Baresi, D. Bianchini, V. De Antonellis, M. G. Fugini, B. Pernici, and P. Plebani, "Context-aware composition of e-services," in *Technologies for E-Services*, Springer, 2003, pp. 28–41.

[115] T. Deng, W. Fan, L. Libkin, and Y. Wu, "On the aggregation problem for synthesized web services," *J. Comput. Syst. Sci.*, 2013.

[116] X. Li, Y. Fan, S. Madnick, and Q. Z. Sheng, "A pattern-based approach to protocol mediation for web services composition," *Inf. Softw. Technol.*, vol. 52, no. 3, pp. 304–323, 2010.

[117] A. Brogi, C. Canal, E. Pimentel, and A. Vallecillo, "Formalizing web service choreographies," *Electron. Notes Theor. Comput. Sci.*, vol. 105, pp. 73–94, 2004.

[118] Y. Du, X. Li, and P. Xiong, "A Petri Net approach to mediation-aided composition of Web services," *Autom. Sci. Eng. IEEE Trans.*, vol. 9, no. 2, pp. 429–435, 2012.

[119] A. Erradi, P. Maheshwari, and V. Tosic, "Policy-driven middleware for self-adaptation of web services compositions," in *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, 2006, pp. 62–80.

[120] L. Ardissono, R. Furnari, G. Petrone, and M. Segnan, "Interaction protocol mediation in web service composition," *Int. J. Web Eng. Technol.*, vol. 6, no. 1, pp. 4–32, 2010.

[121] Y. Taher, M. Parkin, M. Papazoglou, and W. J. van den Heuvel, "Adaptation of Web Service Interactions Using Complex Event Processing Patterns," *Serv.-Oriented Comput.*, pp. 601–609, 2011.

[122] A. Brogi and R. Popescu, "From BPEL processes to YAWL workflows," in *Web Services and Formal Methods*, Springer, 2006, pp. 107–122.

[123] H. H. Lin, T. Aoki, and T. Katayama, "Automated Adaptor Generation for Services Based on Pushdown Model Checking," in *Engineering of Computer Based Systems (ECBS), 2011 18th IEEE International Conference and Workshops on*, 2011, pp. 130–139.

[124] L. De Alfaro and T. A. Henzinger, "Interface automata," *ACM SIGSOFT Softw. Eng. Notes*, vol. 26, no. 5, pp. 109–120, 2001.

[125] M. Dumas, M. Spork, and K. Wang, "Adapt or perish: Algebra and visual notation for service interface adaptation," *Bus. Process Manag.*, pp. 65–80, 2006.

[126] R. Seguel, R. Eshuis, and P. Grefen, "Constructing minimal protocol adaptors for service composition," in *Proceedings of the 4th Workshop on Emerging Web Services Technology*, 2009, pp. 29–38.

[127] Z. Shan, A. Kumar, and P. Grefen, "Towards Integrated Service Adaptation A New Approach Combining Message and Control Flow Adaptation," in *Web Services (ICWS), 2010 IEEE International Conference on*, 2010, pp. 385–392.

[128] E. Kindler, "A compositional partial order semantics for Petri net components," in *Application and Theory of Petri Nets 1997*, Springer, 1997, pp. 235–252.

[129] C. Gierds, A. J. Mooij, and K. Wolf, *Specifying and generating behavioral service adapters based on transformation rules*. Univ., Inst. für Informatik, 2008.

[130] H. R. Motahari Nezhad, B. Benatallah, A. Martens, F. Curbera, and F. Casati, "Semi-automated adaptation of service interactions," in *Proceedings of the 16th international conference on World Wide Web*, 2007, pp. 993–1002.

[131] K. Wang, M. Dumas, C. Ouyang, and J. Vayssiere, "The service adaptation machine," in *on Web Services, 2008. ECOWS'08. IEEE Sixth European Conference*, 2008, pp. 145–154.

[132] E. Wohlstadter, S. Tai, T. Mikalsen, J. Diament, and I. Rouvellou, "A service-oriented middleware for runtime web services interoperability," in *Web Services, 2006. ICWS'06. International Conference on*, 2006, pp. 393–400.

[133] S. Becker, A. Brogi, I. Gorton, S. Overhage, A. Romanovsky, and M. Tivoli, *Towards an engineering approach to component adaptation*. Springer, 2006.

[134] A. Bucchiarone, C. Cappiello, E. Di Nitto, R. Kazhamiakin, V. Mazza, and M. Pistore, "Design for adaptation of service-based applications: main issues and requirements," in *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, 2010, pp. 467–476.

[135] D. Ardagna, M. Comuzzi, E. Mussi, B. Pernici, and P. Plebani, “Paws: A framework for executing adaptive web-service processes,” *IEEE Softw.*, vol. 24, no. 6, p. 39, 2007.
[136] L. Cavallaro and E. Di Nitto, “An Approach to Adapt Service Requests to Actual Service Interfaces,” 2008.
[137] G. Denaro, M. Pezzé, D. Tosi, and D. Schilling, “Towards Self-Adaptive Service-Oriented Architectures,” 2006.
[138] J. Jiang, S. Zhang, P. Gong, and Z. Hong, “Message dependency-based adaptation of services,” in *Services Computing Conference (APSCC), 2011 IEEE Asia-Pacific*, 2011, pp. 442–449.
[139] S. Guinea and P. Spoletini, “Evaluating the compatibility of conversational service interactions,” in *Proceeding of the 3rd international workshop on Principles of engineering service-oriented systems*, 2011, pp. 29–35.
[140] V. Andrikopoulos, S. Benbernou, and M. Papazoglou, “On The Evolution of Services,” *IEEE Trans. Softw. Eng.*, vol. PP, no. 99, pp. 1–1, 2012.
[141] A. Azough, E. Coquery, and M. S. Hacid, “Supporting Web Service Protocol Changes by Propagation,” in *IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology-Volume 01*, 2009, pp. 438–441.
[142] M. Ouederni, G. Salaün, and E. Pimentel, “Client Update: A Solution for Service Evolution,” in *IEEE International Conference on Services Computing (SCC)*, 2011, pp. 394–401.
[143] S. Rinderle, A. Wombacher, and M. Reichert, “On the controlled evolution of process choreographies,” in *Proceedings of the 22nd International Conference on Data Engineering*, 2006, pp. 124–124.
[144] S. H. Ryu, F. Casati, H. Skogsrud, B. Benatallah, and R. Saint-Paul, “Supporting the dynamic evolution of web service protocols in service-oriented architectures,” *ACM Trans. Web TWEB*, vol. 2, no. 2, p. 13, 2008.
[145] S. Wang and M. A. M. Capretz, “A dependency impact analysis model for web services evolution,” in *IEEE International Conference on Web Services, ICWS*, 2009, pp. 359–365.
[146] P. Kaminski, H. Müller, and M. Litoiu, “A design for adaptive web service evolution,” in *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*, 2006, pp. 86–92.
[147] J. Vlissides, R. Helm, R. Johnson, and E. Gamma, “Design patterns: Elements of reusable object-oriented software,” *Read. Addison-Wesley*, 1995.
[148] J. Billington, S. Christensen, K. Van Hee, E. Kindler, O. Kummer, L. Petrucci, R. Post, C. Stehno, and M. Weber, “The Petri net markup language: concepts, technology, and tools,” *Appl. Theory Petri Nets 2003*, pp. 483–505, 2003.
[149] Z. Zhou, L. T. Yang, S. Bhiri, L. Shu, N. Xiong, and M. Hauswirth, “Verifying mediated service interactions considering expected behaviours,” *J. Netw. Comput. Appl.*, 2010.
[150] M. Eslamichalandar, K. Barkaoui, H. R. Motahari Nezhad, “Dynamic Adapter Reconfiguration in the context of Business Protocol Evolution”, *in Proceedings of the the thirteenth IEEE International Conference on Computer and Information Technology (CIT’13)*, 2013.
[151] T. H. Quyet, Q. P. Thi, and D. B. Hoang, “A method of verifying web service composition,” in *Proceedings of the 2010 Symposium on Information and Communication Technology*, pp. 155–162, 2010.
[152] G. Wiederhold and M. Genesereth, “The conceptual basis for mediation services,” *IEEE Expert*, vol. 12, no. 5, pp. 38–47, 1997.

Résumé en Français

Avec l'idée croissante de faire abstraction des technologies sous-jacentes et des implémentations, l'informatique orientée services (SOC) [1] a émergé comme un paradigme de coopération de composants logiciels auto-descriptifs appelés services. Les services sont des composants ouverts et une encapsulation auto-descriptive de fonctionnalités d'entreprise qui soutiennent le développement à faible coût et le déploiement rapide des applications distribuées. Un service a un identifiant et peut fournir sa fonctionnalité via une interface standardisée [2]. La sorte de services la plus importante est celle des services Web [3]. Un service Web tel que défini par le Consortium de World-Wide Web (W3C) est un logiciel identifié par une URI, dont les interfaces et la liaison sont susceptibles d'être défini, décrit et découvert par des artefacts XML, et prend en charge les interactions directes avec d'autres applications logicielles utilisant les messages basés XML via les protocoles basés sur Internet [4]. Les services Web ont été largement adoptés pour mettre en œuvre pratiquement l'idée fondamentale de l'architecture orientée services (SOA) [7-8]. La SOA est une approche du développement d'applications logicielles faiblement couplées et distribuées comme un ensemble de services autonomes bien définis au sein et entre entreprises. La SOA est souvent réalisée par des technologies de services web [10] : les technologies XML, Simple Object Access Protocol (SOAP) [11], Web Services Description Language (WSDL) [12], The Universal Description, Discovery and Integration (UDDI) [13].

La description de service Web représente généralement la capacité de service, le comportement (protocole métier), la qualité, et l'interface du service. Une instance d'un service donné correspond à l'exécution de ses activités. Ces activités sont des unités de travail atomiques où l'ordre partiel d'exécution des activités désigne le protocole métier d'un service. Les résultats attendus et l'approche conceptuelle d'un service signifient sa capacité. La qualité d'un service (QoS) est réalisée par des propriétés non fonctionnelles. Une interface de service représente sa fonctionnalité. L'interface décrit la signature de service (comprenant les

opérations, les entrées / sorties des messages, des types de messages, et les paramètres d'erreur) face à un environnement d'un point de vue interactionnel. L'interface de service est constituée d'un ensemble de portes (reliées par un canal) permettant l'échange de messages pour un service en utilisant SOAP à travers le protocole de transport tel que HTTP ou HTTPS. Les interfaces de services Web sont généralement décrites en utilisant WSDL.

Dans la dernière décennie, de nombreuses approches ont été consacrées à différents aspects des services Web associés à la modélisation, l'interopérabilité, la qualité des services (QoS), la sémantique, la sécurité, et l'automatisation des différents processus tels que la découverte, la sélection, et la composition. Parmi ces défis, notre principale préoccupation dans ce travail est la composition de services. La composition de services Web comme idée fondamentale de l'architecture SOA est de combiner des services individuels et simples dans les processus complexes pour fournir, l'interopérabilité dynamique, automatique et transparente des services.

La composition de services Web a été développée pour soutenir le B2B et l'intégration des applications. La composition de service établit une fonctionnalité à valeur ajoutée par l'intégration de divers services menés par différentes organisations qui ne peut pas être prévue au moment où ces différents service Web sont conçus. Le web service composite est défini récursivement comme une agrégation des web services élémentaires et composites [9].

Il y a deux descriptions différentes de service pour un model de composition : *l'orchestration* de service et la *chorégraphie* de service (également appelé *contrat)* [11]. Bien que toutes les deux soient appliquées au modèle des services composites dans l'architecture SOA, l'orchestration de service décrit l'interaction de service du point de vue d'un service participant simple (*orchestrateur*), tandis que la chorégraphie de service décrit les interactions entre une collection de service participants à partir d'une perspective globale tels que chaque partie sait la logique métier et l'ordre de transmission de messages. Par la chorégraphie, nous nous référons aux échanges de messages qui se produisent entre partenaires dans une interaction de service. Les langages comme BPEL et BPMN décrivent des orchestrations de service. Différents langages de spécification existent également pour la chorégraphie de services tels que WS-CDL [12] et BPEL4Chor [13].

Beaucoup de travaux sur la composition de services Web sont présentés dans la littérature mettant l'accent sur différents aspects de spécifications de services [9-10].

Dans notre travail, nous traitons les interfaces de la couche métier et les protocoles de service Web à un haut niveau d'abstraction de la pile d'interaction de services.

A cause de problèmes des hétérogénéités et de l'autonomie des services Web, il est fondamental d'assurer la correction des services composites. Même si les fournisseurs de services empêchent la publication de services erronés, cependant pour le service client, il est essentiel de vérifier formellement la correction d'un service composite avant de son exécution (en utilisant notamment un outil model-checker). La vérification de la composition de services est inévitable pour éviter une grande perte économique, tout en vérifiant si l'exigence du concepteur et les besoins des utilisateurs sont satisfaits. Les approches de vérification analysent le comportement d'un service composite en se basant sur une variété de critères de conformité et des formalismes différents.

Un des grands défis dans la vérification des services collaboratifs est préoccupé par la compatibilité entre les services participants [14]. Il est crucial pour l'interopérabilité des services Web hétérogènes qu'ils soient compatibles. Deux services sont censés être compatibles si une collaboration entre eux est accomplie avec succès, et aussi si chacun d'eux atteint les résultats prévus (états finaux). Certains travaux de recherche sur les services Web se sont concentrés sur le traitement des incompatibilités d'interaction de service.

Dans cette thèse, nous sommes également motivés sur l'analyse et la vérification du comportement de la composition de services Web. Plus particulièrement, nous voulons assurer l'absence de blocage dans la composition. Généralement, la vérification de l'intégration de services est réalisée en utilisant des techniques basées sur l'espace d'exploration d'état d'un modèle formel d'un service donné.

Nous présentons ici une approche différente, basée sur la théorie de structure de réseaux de Petri [16] permettant la reconnaissance des conditions nécessaires et / ou suffisantes assurant à la fois une composition compatible et une meilleure compréhension des sources d'incompatibilité.

L'objectif principal de notre approche est de montrer comment la théorie de structure de réseaux de Petri peut fournir quelques directives et des solutions pour assurer la justesse de la composition de services Web.

Afin de décrire, modéliser et analyser le comportement des services Web, nous utilisons les open workflow net (oWFN) formalisme [15] qui est une sorte de réseaux de Petri bien adaptées pour faire face à la dimension de flux de contrôle lié à l'évolution du protocole

métier de service. Un oWFN est un type de réseaux de Petri avec des places interface pour la communication asynchrone avec les partenaires.

Malgré d'énormes efforts pour évaluer les critères de compatibilité entre les services [17-19] il y a encore une attention considérable pour vérifier la justesse de la collaboration des services. L'approche typique pour permettre aux services incompatibles d'interagir facilement est l'adaptation de services web. Bien que la normalisation dans les services Web réduit l'hétérogénéité et la rend plus facile leur interopérabilité, l'adaptation des services reste nécessaire. La fonctionnalité d'adaptation peut être offerte pour permettre l'intégration à l'intérieur entre interfaces d'entreprise. La nécessité d'adaptation dans les services Web provient des sources suivantes: assurer l'interopérabilité, l'optimisation, la récupération et le changement de contexte. Principalement, le concept de médiation a été introduit pour les bases de données [20].

Les auteurs de [19] [21-22] identifient les besoins pour l'adaptation dans les services Web en abordant l'hétérogénéité au niveau de *l'interface* de service et le *protocole métier*:

- Les incompatibilités au niveau de l'interface du service comprend les incompatibilités de signature de services (par exemple, les noms du message et de l'opération, le nombre, le type des paramètres entrée / sortie des opérations, et la contrainte de valeur de paramètre) avec les classifications suivantes:
  - Syntaxique. Aucune égalité existe entre le nom des opérations de service et leurs noms de messages d'entrée / sortie. La compatibilité syntaxique assure que l'interface fournie par un service équivalent avec l'interface requise du partenaire et vice-versa.
  - Structurelle. Il existe des différences dans les types attendus ou les valeurs des messages d'entrée / sortie.
  - Sémantique. Il existe des différences dans l'interprétation de la signification d'un élément de données ou la fonction d'une opération.
  - Messages division/ fusion. Un seul message d'un service correspond à plusieurs messages dans un autre service de la même fonctionnalité, ou plusieurs messages d'un service ont une même correspondance dans un autre.
- Les incompatibilités au niveau du protocole métier (ou le comportement du service) sont concernés par les dépendances d'échange de messages entre les services (par exemple, des blocages où les deux services partenaires attendent mutuellement de

recevoir un message de l'autre, et la réception non spécifiée dans lequel un service envoie un message tandis son partenaire ne s'y attend pas):

- Contrainte de commande. La contrainte que les services imposent sur les séquences d'échange de messages.
- Les messages supplémentaires / manquants : Un service délivre un message qui n'est pas spécifié dans un autre partenaire de service et vice versa.

De nombreuses approches d'adaptation ont été proposées pour faire face à la fois aux incompatibilités de l'interface de service et du protocole métier entre les fonctionnalités fournies et requises de services développés par des parties différentes [23-29]. Les approches proposées s'appuient sur l'une de ces deux techniques: la modification de service ou la synthèse d'un composant adaptateur. L'adaptation en termes de modification de service exige l'application de certaines mesures de réglage pour supporter les spécifications du service partenaire. Alors que dans l'interaction de service où l'adaptation s'occupe de la création d'un adaptateur, un composant autonome médiatise les interactions entre les deux services avec des interfaces et des protocoles potentiellement différents de telle sorte que l'interopérabilité est atteinte. Le problème de la synthèse des adaptateurs pour l'interaction de services incompatibles a été étudié dans le domaine de la SOA ainsi que dans le domaine du génie logiciel à base de composants.

Un composant adaptateur est ajouté entre les services partenaires que son rôle principal est de faire des correspondances entres les messages ou gérer les échanges entre des derniers. L'utilisation d'un tel médiateur nous permet la composition correcte entre deux services. Autrement dit, l'adaptateur pour compenser les différences entre les interfaces de services par des fonctions de transformation (par exemple, XSLT, XQuery). L'adaptation de l'interface se présente lorsque l'interface fournit d'un service ne correspond pas à l'interface requise pour être fourni dans une interopérabilité donné. De nombreux outils industriels ont été développées pour l'interface d'adaptation à l'aide des outils de transformation de schéma (par exemple, Microsoft BizTalk Mapper, Stylus Studio de XML mapping tools et SAP Editeur XI Mapping).

En outre, l'adaptateur réconcilie les incompatibilités entre les protocoles métier de service en réorganisant les échanges de messages ou la génération d'un message manquant [30]. Particulièrement, certaines des approches d'adaptation se sont concentrées sur la

réconciliation des incompatibilités entre les interfaces comportementales, dans lequel les interfaces capturent les contraintes l'ordre entre les échanges de messages. Dans cette thèse, nous avons examiné les différentes approches d'adaptation proposées pour faire face aux incompatibilités dans collaboration de services Web.

Ici, nous nous concentrons sur l'adaptation des services du point de vue de leurs protocoles métier et de résumer de tous les autres aspects tels que les propriétés non-fonctionnelles (par exemple, la contrainte de temps, coût), les données et la sémantique de l'information.

Évidemment, l'analyse de compatibilité [31] et les techniques d'adaptation pour faire respecter l'interopérabilité des services ne sont pas de nouvelles idées. Le principal défi de cette thèse est de se concentrer sur l'adaptation de service en particulier, dans le cas où les protocoles métier de services sont engagés dans une l'interopérabilité évolutive.

Dans tous les travaux de l'adaptation de service, le protocole métier est supposé qu'il ne change pas.

Les approches d'adaptation de service existantes prennent en considération que les protocoles métier ne changent pas après la génération d'un adaptateur. Cependant, il existe deux cas où les protocoles métier service changent au cours des interactions de services: l'une est que la plupart des services du monde réel fournissent des modèles d'interaction pour un ensemble d'activités connexes, telles que dans une session d'interaction, deux services peuvent s'engager dans des conversations qui comprennent deux ou plusieurs de ces modes d'interaction. Un exemple de ces services est Google Checkout. L'autre scénario, c'est quand les protocoles métiers d'un service évoluent. Sur cette préoccupation, dans la deuxième partie de la thèse, nous nous concentrons sur le problème du service d'adaptation lorsque les protocoles métier évoluent à une nouvelle version. Les défis comprennent l'identification des changements sur les protocoles métiers, et leurs impacts respectifs sur les adaptateurs.

Une telle analyse devrait distinguer les changements qui n'ont pas d'impact sur l'adaptateur, ou ils peuvent être adaptées dans avec changement dynamique de la spécification de l'adaptateur par des approches telles que les logiciels à base aspect [33], ou ils exigent totalement une nouvelle génération de la adaptateur. Toutefois, la mise à jour de la description de l'adaptateur au moment de l'exécution peut être préférable à une régénération complète. La finalité est de fournir suffisamment d'informations à l'adaptateur développeurs pour décider comment faire face à des changements dans les services interactifs sous-jacents.

Pour de diverses raisons, les fournisseurs de services peuvent changer leurs protocoles métiers. Un défi intéressant pour l'adaptation de service afin de faire face à l'évolution dynamique de protocole P d'un service donné (par exemple, P est modifié à P') qui est adapté par un adaptateur. L'évolution des services Web est généralement entraînée par une motivation perfective - de modifier la fonctionnalité existante de services ou de règles métiers; et également par une motivation correctives - changer la signature de service, le comportement ou la politique [32]. L'application de ces modifications entraîne des défis dans de nombreux aspects de la gestion des changements dans les adaptateurs. La question principale est ce qui se passe au développement de l'adaptateur dans ces cas.

Évidemment, la construction d'un nouvel adaptateur à partir de zéro à chaque fois que le protocole métier évolue est parfois coûteuse. Par conséquent, nous préférons les approches qui mettent à jour les spécifications de l'adaptateur que pour les pièces de protocoles qui ont été modifiés plutôt que total exploration d'espace des états _ du modèle composé (c'est à dire, l'adaptation à partir de zéro), si c'est logique. Par conséquent, pour obtenir une réponse raisonnable à ces questions d'ambiguïté, nous cherchons à mettre l'accent plus sur eux. Par conséquent, nous nous intéressons à fournir une technique pour évaluer automatiquement, à quel point un protocole de service métier a changé, ou à quel point reste sans changement et par la suite à quel point une interaction doivent être traitées pour l'adaptation aux changements. Par conséquent, notre méthode doit déterminer le mécanisme d'adaptation qui doit être à nouveau réalisée pour ces interactions affectées avec la nouvelle version des protocoles de service. Cette technique doit tenir compte de ces éléments d'un protocole adaptateur qui doit être réarrangé après ces changements. Cependant, dans certains cas, l'évolution de protocole métier est plus efficace pour réaliser la génération de l'adaptateur à partir de zéro.

Une question intéressante est de comprendre l'impact des changements sur la spécification de l'adaptateur existant. Une approche de base serait de régénérer l'adaptateur à partir de zéro chaque fois qu'il y a un changement sur les protocoles métiers des services partenaires. Une telle approche n'est pas efficace dans le cas où les changements de protocoles d'affaires n'ont pas d'impact global. Au lieu de cela, une approche plus bénéfique est d'analyser les impacts potentiels de ces changements sur l'adaptateur supportant les interactions entre les services. Par conséquent, il y a une nécessité d'une méthode pour détecter l'évolution des protocoles

métier participant à un adaptateur, et d'identifier si la régénération d'un adaptateur est nécessaire, ou encore les changements peut être corrigé à la volée.

Nous présentons d'abord une méthode pour identifier des changements dans des protocoles de services participants dans un adaptateur.

La méthode proposée permet de détecter les éléments d'un protocole métier donné qui ont été modifiés, ie.,, des messages ou des activités ajoutés, supprimés ou mis à jour. Ensuite, nous identifions les impacts potentiels des changements sur l'adaptateur courant soit en termes de l'impact partiel (changements, dynamique traitables) ou de l'impact global (ie., les changements qui nécessitent re-génération d'adaptateur à). Pour que les modifications avec un impact partiel, nous trouvons et nous suggérons de mettre à jour les spécifications de l'adaptateur courant dynamiquement. Nous vérifions également la justesse du nouveau adaptateur qui est dynamiquement re-configuré. En particulier, nous faisons les contributions suivantes qui résument les principaux points de notre travail:

- **Introduction des Patterns d'adaptation pour un adaptateur.** Nous identifions et classons les paternes communs d'adaptation possibles (AAP) pour évolution des protocoles métier. Pour ce but, nous abordons la taxonomie de ces AAP par rapport au type de changements qui peuvent survenir au niveau des éléments de l'interface de services (par exemple, des messages ou des activités ajoutés, supprimés ou mis à jour en différents points des interactions). Par conséquent, chacun des AAP caractérise un paterne d'évolution du protocole métier (_, le type de changement) et décrit l'impact potentiel du paterne d'évolution à l'adaptateur, et qui peut être un impact partielle ou globale. Une AAP recommande également une stratégie pour la mise à jour de la spécification de l'adaptateur à la volée, si l'impact du pattern d'évolution respective est partiel et une telle solution existe.
- **L'identification des modèles d'évolution du protocole.** Nous présentons un algorithme pour détecter les changements de protocoles d'affaires en termes de modèles (des PAA) de l'évolution. L'algorithme permet de reconnaître les éléments de protocole métier qui ont été changé en termes d'éléments de l'interface ajoutées, supprimées et mis à jour.
- **Protocole Evolution étude d'impact sur l'adaptateur.** Nous présentons un algorithme pour analyser automatiquement l'impact de l'évolution dans les protocoles métiers en vérifiant la AAPs. L'algorithme d'analyse d'impact explore les zones affectés du protocole métier en utilisant une méthode basée bfs (recherche en largeur d'abord) et évalue les

influences potentielles des changements en vérifiant les AAPs correspondantes. Pour chacun des AAP impliqué avec un impact partiel, l'algorithme applique dynamiquement les stratégies d'adaptation de l'adaptateur courant pour re-configurer la spécification d'adaptateur. Sinon, l'algorithme detecte que l'adaptateur doit être complètement reconstruit à zero -en utilisant une analyse globale des nouvelles interactions entre les services pour les protocoles métiers évolué.

- **Vérification de l'adaptateur.** Nous utilisons une technique basée sur la théorie de la structure des réseaux de Petri [28] pour vérifier dynamiquement la justesse de l'adaptateur à quel point est actualisé à la volée. En effet, il est essentiel de vérifier la bonne formation [30] la propriété de l'adaptateur généré. Un adaptateur est dit être bien formé s'il supporte une collaboration compatible entre les services partenaires. Nous montrons comment nous pouvons prendre les avantages de concepts basés sur la théorie de la structure des réseaux de Petri pour vérifier si le nouveau adaptateur satisfait la propriété de la bonne formation.
- **Implémentation du prototype et expérimentations.** Enfin, nous présentons un prototype qui est un outil d'application de l'approche proposée basée sur la PIPE2 (plate-forme indépendante de Petri Editor 2) [34]. Les résultats expérimentaux obtenus montrent que les développeurs de l'adaptateur permet d'économiser beaucoup de temps, le coût et les efforts en appliquant cette approche.

Au meilleur de notre connaissance de l'état de l'art, ce travail est le premier à faire face à la reconfiguration dynamique de l'adaptateur dans le contexte de l'évolution des protocoles métier de services partenaire. L'approche proposée est utile pour les clients de services avec des adaptateurs en place quand ils reçoivent des avis de changement dans les protocoles métiers partenaires afin d'analyser l'impact et assurer la compatibilité, et d'identifier d'éventuelles mises à jour de la spécification d'adaptateur pour remédier aux changements. Par conséquent, d'avoir une telle méthode d'analyse sensible aux évolutions, ils peuvent efficacement faire un adaptateur approprié au moment de la conception par rapport aux méthodes régénération d'adaptateur. Un autre résultat de ce travail est d'identifier le type de l'évolution du protocole métier permettant aux développeurs de l'adaptateur d'éviter régénération complète de l'adaptateur.

Comme nous l'avons déjà précisé, dans notre travail, nous ne considérons pas les exigences non fonctionnelles des services Web, un service Web peut donc être considéré comme une structure de contrôle décrivant son comportement en fonction d'une interface pour communiquer de manière asynchrone avec d'autres services.

le cnam

Maryam ESLAMICHALANDAR
Compatibilité de la composition de services Web: Adaptation suite à l'évolution des protocoles métier

Résumé

Avec l'utilisation croissante d'architectures logicielles indépendantes de la plate-forme et du langage dans le paradigme de *l'architecture orientée services* (SOA), la technologie de services web permet l'interopérabilité dynamique et flexible des processus métiers aussi bien au niveau intra qu'inter-organisationnel. Bien que la normalisation des services web permet de réduire l'hétérogénéité et rend plus facile leur interopérabilité, il y a toujours besoin de vérifier leur compatibilité en particulier dans le contexte inter-entreprises. Deux services sont compatibles si une collaboration entre eux est accomplie avec succès et que chacun puisse atteindre ses résultats attendus (états finaux). L'approche typique devant permettre à des services incompatibles d'interagir correctement est *l'adaptation* du service. L'adaptation consiste dans ce contexte à faire face principalement aux discordances relevées au niveau des *interfaces* de service (incompatibilités entre signatures de services) ainsi qu'aux discordances qui ont lieu au niveau des *protocoles métiers* (incompatibilité dans l'ordre des messages échangés entre services). On distingue deux principales techniques d'adaptation: modification de service ou synthèse d'un composant adaptateur. L'adaptation en termes de modification de service exige l'application de certaines mesures d'optimisation pour supporter les spécifications du service partenaire. Dans le cas où l'adaptation traite de la création d'un adaptateur, un composant autonome modère les interactions entre les deux services de sorte que l'interopérabilité soit obtenue. En d'autres termes, l'adaptateur compense les différences entre interfaces de services par conversion de données (c'est-à-dire par transformation de message) et celles entre protocoles métiers en réorganisant les échanges de messages ou en générant un message manquant.

Nous nous concentrons ici sur le problème de la reconfiguration dynamique de l'adaptateur en presence d'évolution de protocols métiers. Après avoir traité de la vérification d'un adaptateur en exploitant des techniques structurelles existantes développées dans le cadre de la théorie des réseaux de Petri, nous établissons une identification des patrons de mise à jour d'adaptateurs ainsi que la mise en correspondance de ces patrons avec les différents types d'évolutions possibles au niveau des protocoles métiers des services web. Ce travail a abouti à la proposition d'un algorithme permettant, d'une part de détecter les patrons d'évolution adéquats suite à une évolution d'un des protocoles métier des services partenaires et, d'autre part et sous certaines conditions, la mise à jour à la volée de la specification du nouvel adaptateur obtenu ainsi que sa verification. Enfin, les expérimentations réalisées sur un prototype montrent les avantages en termes de temps et de coût de l'approche dynamique proposée par rapport aux méthodes statiques conduisant systématiquement à la regeneration complète de l'adaptateur.

Mots-clés: Services Web, composition de services, compatibilité, évolution de protocole métier, adaptation.

Abstract

The advent of Web service technologies in the paradigm of *Service oriented architecture* (SOA) enables dynamic and flexible interoperation of distributed business processes within and across organization boundaries. One of the challenges in working with heterogeneous and autonomous Web services is the need to ensure their interoperability and compatibility. The typical approach for enabling incompatible services to interact is service *adaptation*. The need for adaptation in Web services comes from the heterogeneity at the levels of service *interface* and *business protocol*. The service interface incompatibilities include service *signature* mismatches (e.g., message and operation name, number; the type of input/output message parameters of operations; and the parameter value constraint). The mismatches at the business protocol (or service behavior) level arise from the order constraints that services impose on messages exchanges (e.g., *deadlock* where both partner services are mutually waiting to receive some message from the other, and *unspecified reception* in which one service sends a message while the partner is not expecting it). In service interaction through adaptation, an *adapter* mediates the interactions between two services with potentially different interfaces and business protocols such that the interoperability is achieved, i.e., adapter compensates for the differences between their interfaces by data mappings, and between their business protocols by rearranging the messages exchanges or generating a missing message.

In this dissertation, we focus on how to cope with the dynamic evolution of business protocol P of a given service (i.e., P is changed to P') that is adapted by an adapter in the context of service interaction. Web service specifications constantly evolve. For variety of reasons, service providers may change their business protocols. Therefore, it is important to understand the potential impacts of the changes arising from the evolution of service business protocol on the adapter.

We present an approach to automatically detect the effects of business protocols evolution on the adapter and, if possible, to suggest fixes to update the specification of adapter on-the-fly. Besides, we propose a technique to verify the correctness of new adapter which is dynamically re-configured. Finally, we describe a prototype tool where experimentations show the benefits of proposed approach in terms of time and cost compared to the static methods aiming for complete regeneration of adapter or manual inspection and adaption of the adapter with respect to changes in the business protocols.

Keywords: Web services, compatibility, service composition, business protocol evolution, adaptation.