Modeling and Verification of Web Services Using Recursive Composition Algebra

A thesis submitted to University of Hyderabad in partial fulfillment
for the degree of

Doctor of Philosophy

by

Gopal Narayan Rai

School of Computer and Information Sciences
University of Hyderabad
Hyderabad – 500046
Telangana, India

June 2017

CERTIFICATE

This is to certify that the thesis entitled **"Modeling and Verification of Web Services Using Recursive Composition Algebra"** submitted by **Gopal Narayan Rai** bearing **Reg. No. 12MCPC05** in partial fulfillment of the requirements for the award of **Doctor of Philosophy** in **Computer Science** is a bonafide work carried out by him under my supervision and guidance at IDRBT, Hyderabad.
This thesis is free from plagiarism and has not been submitted previously in part or in full to this or any other University or Institution for award of any degree or diploma.
Parts of this thesis have been published in the following publications:

1. Architectural characterization of Web service interaction verification (Ch. 1).
2. Algebraic modeling and verification of Web service composition (Ch. 3).
3. Set partition and trace based verification of Web service composition (Ch. 4).

Further, the student has passed the following courses towards fulfillment of coursework requirement for Ph.D.

	Course Code	Name	Credits	Pass/Fail
1	CS801	Data Structure and Algorithms	4	Pass
2	CS802	Operating System and Programming	4	Pass
3	CS821	Trends in Software Engineering	4	Pass
4	CS810	Advanced Networking	4	Pass

Supervisor	Director	Dean
Dr. G. R. Gangadharan	Dr. A. S. Ramasastri	Prof. Arun Agrawal
IDRBT, Hyderabad-500 057	IDRBT, Hyderabad-500 057	SCIS, UoH
India	India	Hyderabad -500 046, India

DECLARATION

I, **Gopal Narayan Rai**, hereby declare that this thesis entitled **"Modeling and Verification of Web Services Using Recursive Composition Algebra"** submitted by me under the guidance and supervision of **Dr. G. R. Gangadharan** is a bonafide research work which is also free from plagiarism. I also declare that it has not been submitted previously in part or in full to this University or any other University or Institution for the award of any degree or diploma. I hereby agree that my thesis can be deposited in Shodganga / INFLIBNET.

A report on plagiarism statistics from the librarian of University of Hyderabad is enclosed.

Date:

Name: Gopal Narayan Rai

Signature of the student
Reg. No: 12MCPC05

//Countersigned//

Signature of the Supervisor:

(Dr. G. R. Gangadharan)

Dedicated To Sambsadashiva, late Dr. Mahil Carr, my teachers, family, & friends

Acknowledgements

I am deeply indebted to **Dr. G. R. Gangadharan**, my thesis advisor, for his guidance, wisdom, and support during the course of my Ph.D. at IDRBT. I am appreciative of the time and effort he has devoted to advising me over the past years, and his consistent encouragement and positive reinforcement have made it a very rewarding experience. I have learnt from him the virtues of rigour and professionalism and I am grateful to him for having kept me focused at the times it mattered a lot.

I would like to thank my doctoral review committee members, **Prof. V. Ravi** and **Prof. Vineet Padmanabhan** for providing invaluable inputs and suggestions ever since I started working on my Ph.D. Their encouragement and insightful comments are truly valuable to my research.

It is my privilege to thank **Dr. A.S. Ramasastri**, Director, IDRBT for reviewing the work and timely suggestions throughout my research. I also thank **Mr. B. Sambamurthy**, Former Director, IDRBT for extending his cooperation at the preliminary stage of my research. I also want to thank **Prof. Arun Agrawal**, Dean, School of Computer and Information Sciences (SCIS), University of Hyderabad, Hyderabad for his academic support throughout research work.

I also want to thank many more people I was lucky to find on my way: Adwitiya, Prashant, Pawan, Deepak, Ghanshyam, Pradeep, Shravan, Deepnarayan, Anup, Chandrasekhar, Dinesh, and Manu among others. Their friendship filled gaps of a lonely period and made me feel “at home" in Hyderabad. I would also like to express my thanks to Vamsi Krishna for helping me in my implementation works.

Finally, my deepest gratitude goes to my family for their unflagging love and support throughout my life. I am indebted to my mother (Smt. Gayatri Devi), father (Sri Bashishth Narayan Rai), gurus (Govind Ji Rai, Sri Sudhakar Rai) for encouraging me to follow my curiosity in science and engineering, as well as teaching me to love, appreciate and give. I would also like to thank my elder sisters (Lalsa, Sindhu, and Suman), elder brother (Govind Ji Rai), younger brothers (Raghav, Madhav, Mrityunjay) for their uncountable love. Finally, I would like to thank my wife Vandana especially for her best support and encouragement all times in pursuing my dreams.

Abstract

Service-Oriented Computing (SOC) is a well-established computing paradigm developed over time and still passing through phases of technological refinements day by day. SOC utilizes software services (called as Web services) as fundamental elements for developing and deploying distributed software applications. The design principle of composability among Web services is one of the most crucial reasons for the success and popularity of the services. In the context of Web services, based on the structure, two types of composition are possible: linear composition and recursive composition. In linear composition, the constituent Web services are only basic Web services, whereas in recursive composition, the constituent Web services could be basic as well as composite. The notion of recursive composition requires special attention as it is not easily tractable with classical modeling and verification approaches such as model checking and Petri net.

Designing and running automatic Web service compositions (specifically, recursive ones) are error-prone because a service may have several dependencies with other services to perform its tasks and a developer may not identify the dependee services in advance that would fulfill the request. Interaction among composite services through asynchronous messages opens the space for concurrency related bugs. Moreover, it is difficult to anticipate behavior of automatic service compositions during execution time, and it is difficult to verify whether they conform to the functional requirement specifications or not. Existing approaches including model checking, Petri net, π calculus, artificial intelligence based algorithms, *Temporal Logic of Actions* (TLA), and case-based reasoning achieve the desired aspects of verification by modeling, planning, and verifying the Web services. However, these approaches have their own limitations.

Therefore, in this thesis, we propose a formal modeling technique, namely *Recursive Composition Algebra (RCA)*, to capture the notion of recursive composition among Web services. Application of this algebra on Web services results in a graph, called *Recursive Composition Graph (RCG)*, that works as an interpretation model to verify the various kinds of requirement specifications about Web services. Further, we propose RCA-based verification techniques for the verification of Web services composition, interaction, and compositional equivalence. Given a set of Web services, the composition verification technique verifies behavioral equivalence between every pair of services and finds all possible deadlock conditions. The prime advantage of this technique is that it overcomes the problems associated with the usage of the concepts of the trace equivalence and bisimulation equivalence in the context of Web services. Given a set of Web services and a requirement specification, the interaction verification technique verifies whether the given specification holds or not by generating and analyzing the exhaustive possible interaction patterns for the given set of services. The prime advantage of this technique is that it captures primitive characteristics of Web service interaction patterns, such as recursive dependency, sequential and parallel flow, etc. and it does not require explicit system modeling. Given two Web service composition graphs, compositional equivalence verification technique verifies whether they are compositionally equivalent or not. The prime advantage of this technique is that it reduces the equivalence verification to the subsumption computing between two algebraic expressions instead of using a subgraph matching technique. In addition to the said verification techniques, we provide an integrated tool and a cross-domain application (modeling and analysis of multistage Cyber attacks) of the proposed framework.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Service-Oriented Computing and Web Services

Service-Oriented Computing (SOC) is a computing paradigm that utilizes services as fundamental elements for developing distributed applications in heterogeneous environments. The promise of Service-Oriented Computing is a world of cooperating services that are being loosely coupled to flexibly create dynamic business processes and agile applications that may span organizations and computing platforms, and can adapt quickly and autonomously to the changing requirements [5]. Engineering a service-oriented computing system is a process of discovering and composing the proper services to satisfy a specification, whether it is expressed in terms of a goal graph, a workflow, or some other model [6].

Web Services have become the preferred implementation technology for realizing service-oriented architectures (SOA). The World Wide Web Consortium (W3C) defines a Web service as "A software application identified by a uniform resource identifier (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" [7]. The success of Web services is due to the fact that their development is based on existing, ubiquitous infrastructure such as hypertext transfer protocol (HTTP), simple object access protocol (SOAP), and extensible markup language (XML) [6].

Figure 1.1 shows the Web service architectural model [6] where a service bro-

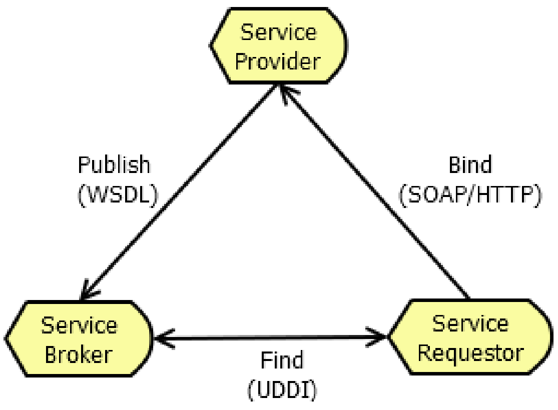


Figure 1.1: Web Services Architectural Model

ker acts as an intermediary between service requesters and service providers. A UDDI-based service registry is a specialized instance of a service broker. Under this configuration, the UDDI registry serves as a broker where the service providers publish the definitions of the services they offer using WSDL, and the service requesters find information about the services available. The design principle of composability among Web services is one of the most crucial reasons for the success and popularity of the services. Service composition is perceived as the federation of a service with other remote services, specifying the participating services, the invocation sequence of services and the methods for handling exceptions [8].

1.2 Modeling and Verification of Web Services

Designing and running Web services compositions are error-prone because a single service may have several dependencies with other services to perform their tasks correctly and the developers may not know the identity of those services that would fulfill the request. Analogous to other distributed systems based on asynchronous communication, it is difficult to anticipate how Web service compositions behave during execution and whether they conform to the functional requirements [9]. Their composition and conversation might lead to concurrency-related bugs, security threats, undesired communication patterns, poor qualities of service, etc. Therefore, verification of Web services is unavoidable. Following are the various classes of formal techniques being used for modeling and verification of Web services.

1.2.1 Model Checking Based Modeling and Verification

Model checking [10] is a popular formal technique facilitating automatic verification of finite-state transition systems, and it has been applied for almost all of the Web services verification aspects, such as control-flow [11][12], data-flow [13], interaction [14][15], time requirements [16][17], quality of service [18][19], security requirements [20], etc. Model checking verifies the desired behavioral properties of Web services for a given model (say, choreography or orchestration) through complete exploration of all the reachable states and the properties that disseminate through them. In model checking, temporal logics (for instance, linear temporal logic (LTL) and computation tree logic (CTL)) are used for specifying the properties of the given model. A temporal logic uses atomic propositions and Boolean connectives to build up complicated expressions describing the properties of the reactive Web service model. Various flavors of model checking with their accompanying tools are available, such as bounded model checking, probabilistic model checking, abstract model checking, timed model checking, etc.

1.2.2 Petri Net Based Modeling and Verification

Petri net is a well-established process-modeling approach [21]. A Petri net is a directed, connected, and bipartite graph in which nodes represent places and transitions, and tokens occupy places. It has contributed greatly to the development of a rich theory of concurrent systems. Moreover, their ease of conceptual modeling (largely due to an easy-to-understand graphical notation) has made Petri nets as the model of choice in many applications. Petri nets are very popular in Web service related fields because they can capture [22] the large varieties of process control flows. In particular, the dead-path-elimination technique that is used in BPEL to bypass activities whose preconditions are not met, can be readily modeled in Petri nets [23]. Apart from this, Petri net is used for analyzing compatibility between services [24, 25], composition verification [26], timing constraints verification [27], etc.

1.2.3 Artificial Intelligence Based Modeling and Verification

Artificial intelligence (AI) based techniques emphasize the creation of intelligent systems that work and react like humans in the context of learning, planning, and problem solving. The problem of automatic Web service composition generation is closely related to the problem of *Goal-Oriented Action Planning (GOAP)* in artificial intelligence [28, 29, 30]. In literature, several AI planning based techniques for automatic composition are available: STRIPS-based [29], PDDL-based [31], HTN-based [32], etc. McIlraith and Son [33] adopted and extended Golog to allow the automatic building of Web services. PDDL is a widely used formal language to describe different kinds of planning problems. Hierarchical planning is an AI planning methodology that creates plans by task decomposition. Well-known hierarchical planners SHOP and SHOP2 (Simple Hierarchical Ordered Planner) have been used for Web services [34].

1.2.4 Process Algebra Based Modeling and Verification

A process algebra [35] is a concurrent language that abstracts many details and focuses on particular features. Process algebras are precise and well studied formalisms that allow the automatic verification of both functional and non-functional properties. They come with a rich theory on bisimulation analysis, i.e. to establish whether two processes have equivalent behaviors. Such analyses are useful to establish whether one service can substitute another service in a service composition [36] or to verify the redundancy of a service. Algebraic service composition aims to introduce much simpler descriptions than other approaches, and to model services as mobile processes to ensure verification of properties [37, 38, 39, 40] such as safety, liveness, and resource management.

1.3 Primitive Characteristics of Web Services from the Modeling and Verification Perspective

Though modeling and verification of Web service composition is a well-explored research area, there is a paucity of widely accepted standard solutions that provide error-free, automatic, and dynamic service composition. Although the existing solutions are promising, core techniques adopted in the solutions do not capture all the required characteristics of Web service interaction verification as they were proposed natively for different scenarios and applications. For instance, Symbolic Model Verifier (SMV) [5] is an input language for model checking that does not support modeling subtleties regarding Web service interaction such as automatic discovery of services and dynamic availability of Web services [6]. The primitive characteristics of the Web services that need to be well considered from the modeling and verification perspective are discussed as follows.

1.3.1 Linear and Recursive Composition

A Web service can compose with other services in order to realize a business logic, where invocation order of candidate services are explicitly mentioned and exception handling methods are provided. Linear Composition and recursive composition [41, 42, 43] are the fundamental characteristics of the Web services. Let us consider that $\mathcal{W}_B$ is a finite set of basic Web services and $\mathcal{W}_C$ is a finite set of composite Web services. Then, **linear composition** refers a composition scenario where every member of $\mathcal{W}_C$ is composed of a non-empty set of Web services (say $\mathcal{W}_S$) such that $\mathcal{W}_S \subseteq \mathcal{W}_B$, whereas **recursive composition** refers a composition scenario where every member of $\mathcal{W}_C$ is composed of a non-empty set of Web services (say $\mathcal{W}_S$) such that $\mathcal{W}_S \subseteq \mathcal{W}_B \cup \mathcal{W}_C$.

1.3.2 Modular and Hierarchical Architecture

All Web services basically fall into two categories: either basic or composite. A basic Web service does not take help of other Web services to accomplish a job, whereas a composite Web service does. A Web service composition scenario has both shades of hierarchical and modular architectural archetypes. It requires modeling a Web service in such a way that the service preserves its identity and supports composite services at a time. A Web service should be able to serve as an independent module, and at the same time, it should be able to serve as a building block for other composite services, if required. **Modular architecture** of a Web service composition model refers to the design of the model composed of independent Web services that could interact with each other [44]. The advantage of a modular architecture is that a module could be added or substituted with another suitable module without affecting the rest of the system. **Hierarchical architecture** of a Web service composition model [45] refers to the design of the model composed of dependent and independent Web services, where a composite service works as a higher (or highest) level abstraction and basic services work as the lowest level abstraction. The advantages of a hierarchical architecture are easy decomposition of the system based on the hierarchy refinement, enhanced scalability, and reuse of the system.

1.3.3 Hierarchical Concurrency

Classical model-based verification approaches do not consider hierarchy among Web services while verifying the requirement specifications. They keep transmitting all variables that are being considered for verification through every state in their state transition diagram, whereas in the context of Web services, all variables need not be considered at a time. A hierarchy must be found among services and must be considered in the verification process. Hierarchical architecture of Web services is discussed in [41, 45], and hierarchical concurrency among Web services is described in [46]. For example, let us consider a scenario depicted in Fig. 1.2, where $\langle A, B\rangle$ represents the knowledge that service A is composed of service B, and $\langle A, B\rangle \rightarrow \langle B, C\rangle$ represents the knowledge that $\langle A, B\rangle$ depends upon $\langle B, C\rangle$. Now, consider that concurrency has to be resolved between B and J for a given specification. If $\langle B, C\rangle$ and $\langle J, K\rangle$ do not affect $\langle B, J\rangle$, then concurrency will be resolved only between B and J without considering C and K. If the concurrency cannot be resolved at the first level (B and J are first level services), then the second level services (C and K are second level services) would be considered. Again, if the second level services are also not sufficient to resolve the concurrency, then the third level services (D and L are third level services) would be considered, and so on.

$$\begin{tikzcd}
\langle A,B \rangle \arrow[d] & & \langle I,J \rangle \arrow[d] \\
\langle B,C \rangle \arrow[d] & \langle B,J \rangle \arrow[l] \arrow[r] & \langle J,K \rangle \arrow[d] \\
\langle C,D \rangle & & \langle K,L \rangle
\end{tikzcd}$$

Figure 1.2: A Web service composition scenario, where $\langle A, B\rangle$ represents the knowledge that service A is composed of service B and $\langle A, B\rangle \rightarrow \langle B, C\rangle$ represents the knowledge that $\langle A, B\rangle$ depends upon $\langle B, C\rangle$. In this scenario, concurrency has to be resolved between B and J

1.3.4 Dynamic Reconfiguration

Hnetynka et al. [47] discussed the dynamic reconfiguration of Web services and collaboration among them. The two crucial factors that make consideration of dynamic reconfiguration in Web services inevitable are: (i) Resemblance of a Web service as a module (a basic Web service resembles an independent module whereas a composite Web service resembles a dependent module), and (ii) Dynamic availability of services. Web services are accessible through the Web and a Web service could become unavailable/removed at any time or a new Web service could be added at any time. Ethically, a composition designer or verifier must be ready for substitution, replacement, and inclusion of services. Inclusion of a new service or unavailability of an existing service could make chaos if not handled properly. Automatic dynamic service composition is a rapidly emerging paradigm and research topic that is based on the concept of dynamic reconfiguration.

1.3.5 Open System Modeling

A Web services workflow module is a reactive system as it consumes messages from the environment and produces answers depending on its internal state [48]. The compositional modeling and design of reactive systems require each component to be viewed as an open system, where an open system is one which is designed to interact with its environment and whose behavior depends on the state of the system as well as the behavior of the environment [49]. A system is termed as an open system if all of its participating entities are not well-known in advance (in terms of their existence and behavior). Similarly, a Web service cannot anticipate the behavior of its interacting partners in advance, thus enabling the open system phenomenon from the perspective of modeling and verification.

1.3.6 Verification of Adversarial Specification

Classical temporal logics such as LTL and CTL are unable to specify collaborative as well as adversarial interactions among different Web services. Alternating Temporal Logic (ATL) [49] is designed to write the requirements of an open system [49] and is able to express collaborative as well as adversarial interaction specifications. ATL models each Web service as an agent. Let Σ be a set of agents corresponding to different Web services, one of which may correspond to the external environment. Then, ATL admits formulas of the form $\langle\langle A \rangle\rangle \Diamond p$, where p is a state predicate and A is a subset of agents. The formula $\langle\langle A \rangle\rangle \Diamond p$ means that the agents in the set A can cooperate to reach a p-state no matter how the remaining agents resolve their choices. ATL is used for Web service verification along with Petri net in [50].

1.3.7 Asynchronous Messaging

A Web service consists ports and a port consists sets of input and output messages. Messages consisting activities are unit of interaction. There are two principal messaging models used in Web services namely synchronous and asynchronous model. The two Web service messaging models are distinguished by their way of request-response operation handling mechanism. Synchronous and asynchronous messaging among less number of services can be handled in a fair manner without much complexity. If the number of participant services are high, asynchronous messaging complicates the verification process.

1.3.8 Trace Modeling

Informally, a trace (in the context of Web services) is a linear, unidirectional Web service composition workflow path in which a node represents a Web service and a directed edge represents the flow from one service to another. Formally, a trace is defined as a tuple $\mathcal{T} = (\mathcal{W}, I, w_i, w_n)$, where $\mathcal{W} = \{w_1, \cdots, w_m\}$ is a finite set of Web services, $I : \mathcal{W} \to \mathcal{W}$ is an invocation function such that $w_i I w_j$ if and only if w_i invokes w_j, $w_i \in \mathcal{W}$ is a service from which the trace begins and $\nexists w_j \in \mathcal{W} : w_j I w_i$, and $w_n \in \mathcal{W}$ is a service on which the trace ends up and $\nexists w_j \in \mathcal{W} : w_n I w_j$. Trace modeling and computation are among the most primitive modeling requirements for Web service interaction as the computation of trace related concepts (such as trace inclusion and trace merging [3]) reduce the time and space complexity for verification. The concept of trace is utilized primarily for studying behavioral equivalence of services.

1.4 Problem Statement and Contributions

On the basis of our study in previous section, we observe that composition among services can take place either in linear or recursive fashion. A Web service composition is a modular-cum-hierarchical architecture, where a service behaves as an open system. Further, we observe that a service composition modeling approach must be able to capture synchronous and asynchronous messaging (among services) in the form of traces, and a verification approach for Web services must support verification of hierarchical concurrency and adversarial specifications. Therefore, we define the problem statement for this thesis as follows:

Let $\mathcal{W}$ be a dynamic (the number of elements in the set may vary with time) set of Web services. Consider that at a time instance t, there are m number of services in the set $\mathcal{W}$ ($\mathcal{W} = \{w_1, \cdots, w_m\}, m \in \mathbb{N}$). Let $\mathcal{W}_B$ be a set of basic services and $\mathcal{W}_C$ be a set of composite services made out of the set $\mathcal{W}$ such that $|\mathcal{W}_B| + |\mathcal{W}_C| = m$. A Web service w_i consists of a set of input messages $w_i.I$ and a set of output messages $w_i.O$ and there is a pre-defined relation from input messages to output messages ($Rl \subseteq w_i.I \times w_i.O$). A Web service $w_i \in \mathcal{W}$ can interact with another Web service $w_j \in \mathcal{W}$ based on input-output compatibility that provides a number of different ways to form a composition. This composition may trigger a sequence of actions and result in various interaction patterns (traces). Out of all possible traces, some traces may be undesirable. Provided a set of available services, our goal is to verify structural and interactive properties by observing *all possible traces while supporting automation in composition formation and dynamism in the set $\mathcal{W}$.*

The contributions of this thesis are: (i) a formal modeling technique, namely *Recursive Composition Algebra (RCA)* to capture the notion of recursive composition among Web services, (ii) a RCA-based verification technique for verifying Web service composition, (iii) a RCA-based verification technique for verifying Web service interaction, and (iv) a RCA-based verification technique for verifying compositional equivalence between Web service composition graphs. Further, we present an integrated tool for Web service verification and a cross-domain application of the proposed technique.

1.5 Thesis Organization

Chapter 2 describes systematic literature review on Web service verification using model checking. In Chapter 3, we propose a formal modeling technique for Web services, namely *Recursive Composition Algebra (RCA)*. Composition verification using RCA is presented in Chapter 4. Technique for interaction verification among Web services using RCA is elaborated in Chapter 5. RCA-based verification technique for verifying compositional equivalence between Web service composition graphs is presented in Chapter 6. A comprehensive tool facilitating our proposed verification techniques is presented in Chapter 7. A multistage Cyber attack modeling and analysis technique using RCA is presented in Chapter 8. Chapter 9 summarizes our contributions including future research directions.

Chapter 2

Model Checking Based Web Service Verification: A Systematic Literature Review

As witnessed by the literature, model checking has been applied for almost all of the Web services verification aspects, such as control-flow, data-flow, interaction, time requirements, quality of service, security requirements, etc. In this chapter, we review the related literature on model checking based Web service verification that can explore: (i) the research motivations behind model checking of Web services, (ii) the existing methods and techniques (with their verification goals) that support model checking of Web services, and (iii) the existing research issues and future research directions in the area of Web service verification.

There are various formal techniques available for the verification of Web services, such as model checking, Petri net, process algebra, etc. However, in this present review, we did not aim for presenting a systematic literature review on verification of Web services that includes all available verification techniques as it would be too broad to cover completely in a chapter. Here, our focus is only on the model checking based Web service verification approaches as model checking is a de-facto verification technique that is applied to almost all of the verification aspects related to Web services. Literature review on other verification techniques for Web services are described in Chapter 4, 5, and 6.

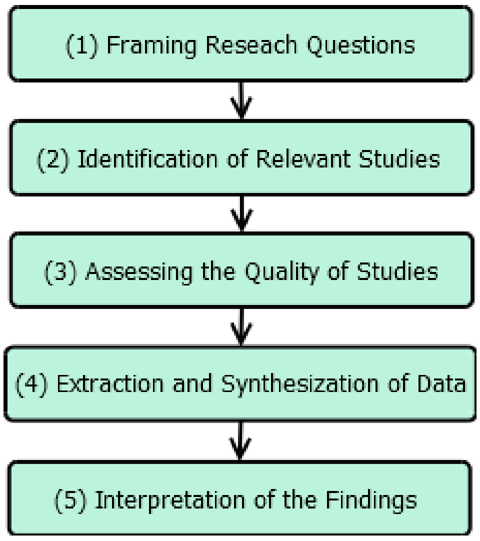

(1) Framing Reseach Questions
(2) Identification of Relevant Studies
(3) Assessing the Quality of Studies
(4) Extraction and Synthesization of Data
(5) Interpretation of the Findings

Figure 2.1: A Five-Step Process of Conducting Systematic Literature Review (Based on [1])

2.1 Review Methodology

A systematic literature review (or simply a systematic review, structured literature review or SLR) is a literature review focused on a research topic that tries to identify, classify, select, and synthesize all high-quality research evidences relevant to that topic area [51]. An SLR is a structured process that identifies, evaluates, and interprets the existing evidences of research that are pertinent to a specific research query. Hence, it follows a predefined search category. An SLR provides a framework for locating new research tasks, discovers gaps in ongoing research, and recommends areas for further examination [51, 52]. We follow a five-step process of conducting systematic literature review as described in [1] (see Fig. 2.1). The process is similar to the methodologies followed in [51, 52, 53].

(1) Framing Research Questions. In order to conduct the review, we frame the research questions and provide their motivations as listed in Table 2.1.

Research Questions	Motivations
RQ1—Why model checking is a popular verification technique for Web services?	To get insight on model checking based Web service verification techniques satisfying functional and non-functional requirements.
RQ2—Which aspects are verifiable using model checking in the context of Web services?	To identify the properties of Web services that are being verified using model checking based techniques.
RQ3—What are the existing approaches, tools, and techniques that help and promote model checking based Web service verification?	For the identification, classification, and comparison of the existing approaches and techniques that are employed in model checking based Web service verification.
RQ4—What are the existing challenges, current research state, and the possible future research directions in the area of model checking based Web service verification?	To identify and explore the potential research challenges that have to be addressed, and to find the possible future research directions in this area.

Table 2.1: Research Questions and Their Motivations

(2) Identification of Relevant Studies. Primarily, we referred articles from the popular online data sources including IEEE-Xplore, ACM Digital Library, Springer, Science Direct, Wiley, IGI Global, Inderscience, and Google Scholar. Non-peer-reviewed articles (such as white papers), editorials, abstracts, short papers (less than 4 pages), and non-English scripts are excluded from the scope of this review. We also did not consider a book or book chapter or dissertation or extension paper or tool paper if the technique and results presented in the document are already published in a journal or conference.

For our search, we provided the search string as follows:

(Web service OR Web services OR service OR BPEL OR BPEL4WS OR choreography OR choreographies OR WSDL OR WS-CDL OR WSFL OR orchestration)

AND

(verification OR verifying OR validation OR checking OR model checking OR UPPAAL OR SPIN OR NuSMV OR PRISM)

(3) Assessing the Quality of Studies. We refined our search using various words like "model checking", "temporal logic", etc. We extracted nearly 582 papers for this review that were published during 2002-2017 in various conferences and journals. The year 2002 was chosen as we did not find a paper on Web service verification published before 2002.

Initial selection. Our initial selection of 582 articles covered the research topic of Web service verification across the search databases. We explored the title, abstract, and keywords of the primarily selected studies and applied the said criterion of inclusion/exclusion.

Final selection. For our final selection, we focused particularly on model checking based Web service verification approaches among the articles collected in initial selection phase and finalized 116 studies for this review.

The fourth step of the systematic review *extraction and synthesization of data* is presented in Section 2.2, and the fifth step *interpretation of findings* is presented in Section 2.3.

2.2 Classification of the Model Checking Based Web Service Verification Approaches

We classify the model checking based Web service verification approaches based on their verification goals classified into six classes: composition verification, behavioral verification, interaction verification, equivalence verification, non-functional requirement verification, and time requirement verification. Further, apart from these classes, there exists research papers on model checking based Web service verification such that they focus on multiple verification goals, and there exists research papers such that they do not belong to any of the said classes. Fig. 2.2 presents a classification of the existing Web service model checking approaches.

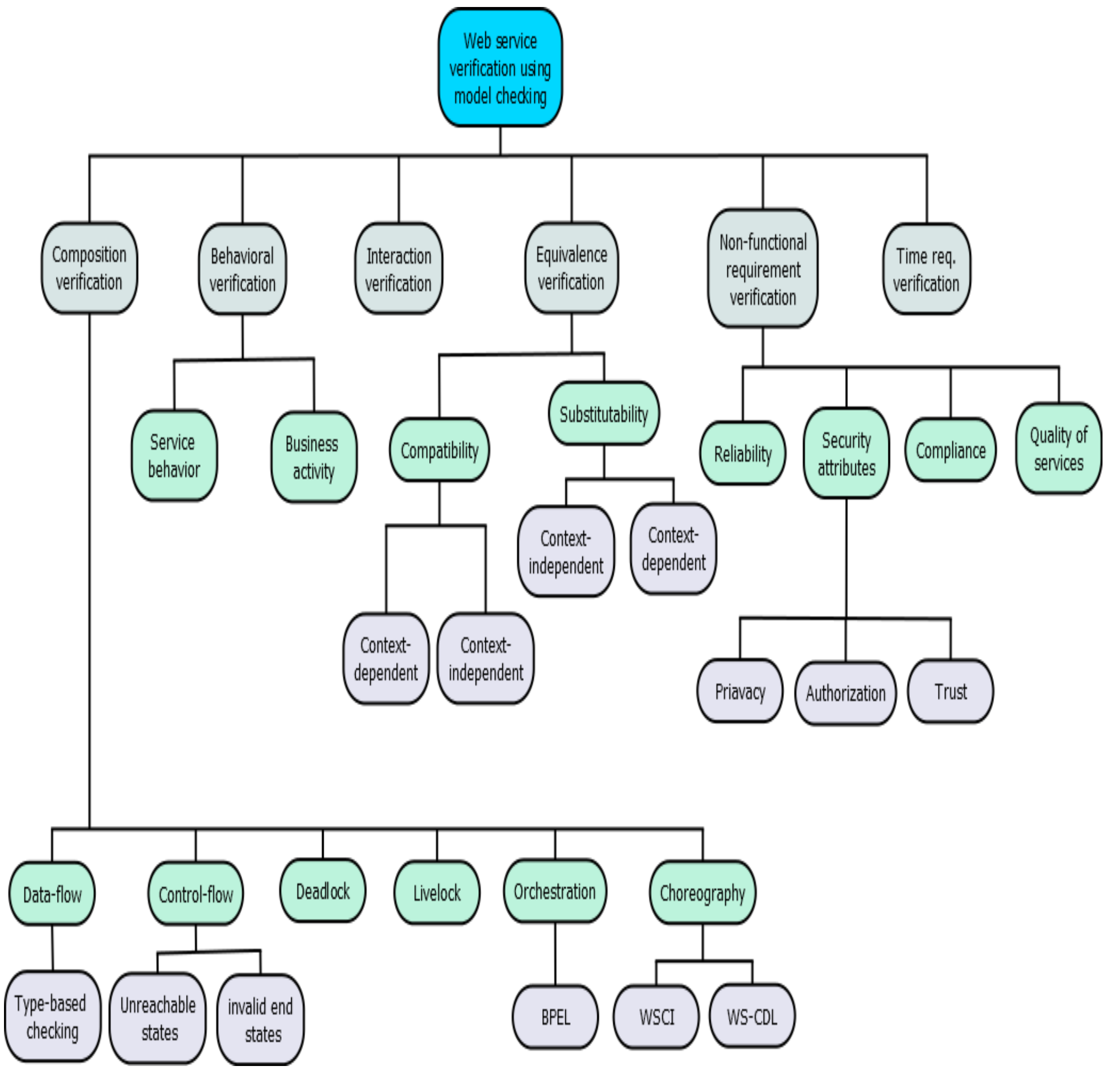


Figure 2.2: Classification of the Model Checking Based Web Service Verification Approaches Based on Their Verification Goal

2.2.1 Web Services Composition Verification

Composition of services specifies the participating services, the invocation sequence of services and the methods for handling exceptions [54]. Huang et al. [55] presented a counter-example driven refinement based model checking technique for Web services. The advantage of their approach is that precision information regarding refinement is distributed among proof slices. Goli et al. [13] proposed a procedure for automated verification of control-flow and data-flow in the *Web Ontology Language* *for Web Services* OWL-S process models by mapping it into equivalent *Process Meta Language* (PROMELA) process definitions. Dai et al. [56] annotated *Business Process Execution Language* (BPEL) and transformed it into time constraints Petri net (TCPN) for the verification of control-flow aspects. Zhang et al. [57] classified the feature interaction into five classes: deadlock, livelock, invocation error, race condition, and resource contention. Corradini et al. [58] presented an approach for the verification of Web service compositions expressed in the *Web Services Choreography Description Language* (WS-CDL) language. Their technique analyzed the given WS-CDL source document and extracted the necessary information. Further, the extracted information was translated to PROMELA code and was verified against the given specification. Qian et al. [59] proposed an automated way to extract formal models from programs implementing Web services using predicate abstraction for abstract model checking.

Dong et al. [60] presented a formalized analysis model that accepts specification and semantics of business flow as inputs and provide asynchronous communication model as an output. Further, this output model is verified by using the Simple Promela Interpreter (SPIN) model checker. Liu et al. [61] presented a model checking based technique for specifying and verifying Web service flow based on annotated OWL-S. In order to verify the Web service flow, similar to [56], they transformed the annotated OWL-S model into a TCPN model. Wang et al. [12] proposed a temporal logic of action (TLA) based composition and verification approach for Web services. Their approach takes OWL-S service description as input and transforms it into TLA description. It produces a composition proposal according to the requests of an user. Luo et al. [62] proposed a model for BPEL based on its grammar and control flow, then they transformed the required part of BPEL into the input language of Model Checking Time and Knowledge (MCTK) and verified the specification properties.

Barker et al. [63] presented a multi-agent protocol (MAP) based Web service choreography language for specifying and verifying service interaction patterns. The advantage with their approach is that services do not have to be pre-configured at design-time. For the verification purpose, they proposed translation from MAP to PROMELA. Xu et al. [64] presented a technique for the verification of Web service choreography written in WS-CDL by mapping it to *Communicating Sequential Processes* (CSP) and using Process Analysis Toolkit (PAT) model checker. Luo et al. [65]

developed a translation procedure to translate BPEL into I/OLTS, and then developed another translation procedure to translate I/OLTS into the input language of ZING model checker. Mei et al. [66] translated BPEL4WS into interface automata, which was then translated into PROMELA. Xinlin [67] proposed a Web service verification approach based on the category theory.

Zhang et al. [68] proposed a *Timed Asynchronous Communication Model* (TACM), which can precisely describe the timed properties and asynchronous communication behaviors. The advantage with TACM is that it can be directly provided as input to UPPAAL model checker. Li et al. [69] proposed a model checking based technique to automatically verify the composite services. For the property specification, they used universal model sequence diagrams (uMSDs). Fu and Chen [70] addressed the problem of state explosion by introducing predicate abstraction and refinement technology in the traditional model checking method. Further, they proposed a framework for Web service composition based on the technology of abstraction and refinement.

Luo et al. [71] proposed a technique for Web service verification that translates BPEL to BSTS and then BSTS to *Interpreted Systems Programming Language* (IPSL) which is input language for the model checker MCMAS. The advantage with their approach is that the epistemic and cooperation properties also can be specified and verified in their framework. Todica et al. [72] proposed a SPIN model checker based technique for verifying automatically generated business process. Zhao et al. [73] proposed a model for guaranteeing correctness of WS-BPEL. The input language for their model is Language Of Temporal Ordering Specification (LOTOS) and they check validity of the model by using model checking. Huang et al. [74] proposed an approach to analyze atomic Web services and extended BLAST model checker to handle concurrency in OWL-S semantics with the use of planning domain definition language (PDDL). Sharygina and Kröning [75] presented an automatic abstraction refinement based model checking technique for Web services. Their technique formalized the semantics of a PHP-like scripting language for implementing Web services by the means of labeled Kripke structures. However, it lacks the support for the verification of liveness properties.

Studies	Verification goal	WS Technology	Model checker
Huang et al. [55]	Trustworthiness of service composition	OWL-S	Blade
Goli and Pathari [13]	Control flow and data flow	OWL-S	SPIN
Dai et al. [56]	Reachability, safety, liveness, and correctness	BPEL	–
Zhang et al. [57]	Detecting feature interactions	BPEL4WS	SPIN
Corradini et al. [58]	WS-CDL choreographies	WS-CDL	SPIN
Qian et al. [59]	Applications that implements services	–	–
Dong et al. [60]	Behavioral specification of the workflow	BPEL	SPIN
Liu et al. [61]	Web service flow	OWL-S	–
Wang et al. [12]	Web service composition	OWL-S	TLC
Luo et al. [62]	BPEL control flow	BPEL	MCTK
Barker et al. [63]	Termination of MAP service choreography	BPEL	SPIN
Xu et al. [64]	WS-CDL based specifications	WS-CDL	PAT
Luo et al. [65]	Correctness of Web services	BPEL	ZING
Mei et al. [76]	Correctness of BPEL4WS	BPEL	SPIN
Xinlin [67]	Control structure of service composition	–	–
Zhang et al. [68]	Asynchronous communication behavior and timed properties	–	UPPAAL
Li et al. [69]	Correctness of composite services	BPEL	SPIN
Fu and Chen [70]	Web service composition	–	SPIN
Luo et al. [71]	Web service composition	BPEL	MCMAS
Todica et al. [72]	Business Processes	BPEL	SPIN
Zhao et al. [77]	BPEL-based service composition	BPEL	CADP
Zahoor [11]	Declarative service composition processes	–	–
Huang et al. [74]	Concurrency in OWL-S	OWL-S	BLAST
Sharygina and Kröning [75]	Concurrency, deadlock, and livelock	–	–
Yu et al. [78]	Concurrency in BPEL	BPEL	–

Table 2.2: Summary of Different Studies on Web Service Composition Verification Using Model Checking

Yu et al. [78] presented a modular verification method for composite Web services which were written in BPEL. Their focus was on the interleaving semantics of BPEL and concurrently running processes. They used the symbolic encoding for modeling the processes. Table 2.2 presents a summary of different studies on Web service composition verification using model checking.

2.2.2 Web Services Behavioral Verification

Behavioral specification of a service composition expresses a causal relationship among various invocations by using control and data flow links [46]. Dìaz et al. [79] presented the unification of WS-CDL and BPEL documents. Their aim was to generate the correct BPEL skeleton documents from the WS-CDL documents by using the timed automata as an intermediary model in order to check the correctness of the generated Web Services with the model checker UPPAAL. Yeung [80] presented mappings of WS-CDL and BPEL into CSP for specifying and verifying the behavior of Web services based business processes. Verification process can be carried out based on the notion of trace-refinement of CSP and can take the advantage of FDR2 model checker. Nakajima [46] proposed an approach to extract the behavioral specification from a BPEL application program and to analyze it by using the SPIN model checker. This approach considers dead-path elimination (DPE) and the abstraction of control variables.

Wang et al. [81] proposed a method for verifying the composed service against certain requirement specifications. Temporal logic of actions (TLA) is introduced to enable effective verification. Specifically, algorithms to transform Web services from OWL-S to TLA were presented. Further, it was shown that automatically verifying Web service behaviors can be achieved by using the model checker TLC. Hallé [82] proposed a method for generating Web service stubs by using the first-order temporal logic, called LTL-FO+. In this approach, the model checker NuSMV is used as a back-end engine to produce message traces compliant with the requirement formulae specified in LTL-FO+. Bentakouk et al. [83] proposed a formal testing framework for addressing the issue of behavioral verification of service composition at design time. The proposed approach is based on the symbolic testing and an SMT solver.

Gao et al. [84] used live sequence chart specifications (LSC) to specify the complex behaviors among multiple Web services, and then translated LSC to extended labeled transition system (ELTS). They proposed a projection approach to check the behavioral correctness and consistency of functional requirements against the ELTS model using NuSMV. Dìaz et al. [85] presented a timed automata based generic model for publishing and managing Web service resources that includes operations for clients to discover and subscribe to resources, with the intention of being notified when the resource property values fulfill certain conditions. Further, they provided an error handling mechanism to deal with discovery failure and resource life expiration, and used UPPAAL to verify the model soundness. Oghabi et al. [86] proposed a formal approach for automating web service verification. They used OWL-S to describe Web services behavior. They parsed the OWL-S file and automatically transformed it into a corresponding Markov chain diagram or Markov decision process, which are then transformed into a PRISM model and provided to the PRISM model checker for verification. Table 2.3 presents a summary of different studies on Web service behavior verification using model checking.

Studies	**Verification goal**	**WS Technology**	**Model checker**
Dìaz et al. [79]	Correct generation of BPEL	WS-CDL	UPPAAL
Yeung [80]	Behavioral specification	WS-CDL, BPEL	FDR2
Nakajima [46]	Behavioral specification	BPEL	–
Wang et al. [81]	Service behavior	OWL-S	TLC
Hallé [82]	Service behavior	–	NuSMV
Bentakouk et al. [83]	Behavioral conformance	BPEL	–
Gao et al. [84]	Behavioral correctness	–	NuSMV
Azaiez and Sbaï [87]	Behavioral properties	WS-CDL	NuSMV
Dìaz et al. [85]	Publish-subscribe architecture	WSRF	UPPAL
Ravn et al. [88]	Business activity	–	UPPAAL
Oghabi et al. [89]	Business activity	OWL-S	PRISM
Marques et al. [90]	Business activity protocol	–	UPPAAL

Table 2.3: Summary of Different Studies on Web Service Behavior Verification Using Model Checking

2.2.3 Web Services Interaction verification

In a Web service composition scenario, an interaction pattern is a description of the overall interactions of candidate services in the composition. The interactions are modeled as links between endpoints of the Web Services' interfaces, each link corresponding to the interaction of one Web Service with an operation of another Web Service's interface. Application of model checking based techniques for the verification of Web service interaction is bit old, but still is in use because of its efficacy. Foster et al. [91] proposed a model-based technique to verify Web service compositions represented in the form of BPEL. They modeled specifications in the form of message sequence charts (MSCs). Further, BPEL and MSCs were mechanically compiled into the finite state process notation (FSP). Then, verification process takes place between FSPs generated from BPEL and MSCs using trace equivalence. Contrary to [91], Fu et al. [92] presented a Web service interaction verification scheme based on the centralized theme of conversation modeling. They specified the desired conversations of a Web service as a guarded automaton. Their focus was on the asynchronous messaging and they made effort to relax the restrictions in the way of direct application of model checking.

Kazhamiakin et al. [93] presented a specification and verification technique for Web services. They used formal tropos for specification modeling and PROMELA for process modeling. Walton [94] verified the interaction among agents participating in multi-agent Web service systems by proposing a Web service architecture and a lightweight protocol language. Further, he verified the specification properties written in the proposed language using model checking. Fu et al. [95] presented a realizability analysis framework that consists of three steps: (i) specification of a service composition using a realizable conversation protocol, (ii) verification of the desired LTL properties on the conversation protocol, and (iii) projection of the conversation protocol to each pair. Pistore et al. [96] presented a requirements-driven design and verification approach for Web services. They used a model checking based technique for validating specifications and verifying requirements. Techniques presented in [91, 92, 94] were efficient, however, they did not deal with the automation of verification process.

Cao et al. [97] presented a model-based verification framework for BPEL. BPEL is modeled by using UML activity diagram and verified using software model checking technique. Dìaz et al. [98] presented an analysis technique for the timed behavior of Web services. Their technique is based on the translation of service description into the timed automata, and then, they used UPPAAL model checker to simulate and analyze the system behavior. Zheng et al. [99] presented a test case generation framework for BPEL using SPIN and NuSMV. They modeled BPEL as Web service automata (WSA) and on the basis of WSA, they presented their test case generation framework. Test cases were used to verify whether the implementation of Web services meet the pre-specified BPEL behavior. Deutsch et al. [100] presented a technique to verify the sequence of events (input, states, and action) resulting from the interaction of Web services in data-driven web applications.

Gu et al. [101] presented a method for verification of Web service conversations written in WSCL by using SPIN. Parizek and Adamek [102] presented a verification technique for checking the compliance of a composite service with the constraints on the order of operation invocations on stateful basic services. Shegalov and Weikum [103] described the generic recovery protocols in the interaction contracts framework that guarantees exactly-once execution and applied model checking to prove its correctness. Wan et al. [104] presented a multi-agent based verification technique for Web services, where Web service communities are modeled as UML activity diagrams. Specifications are expressed in CTL* and MCMAS model checker is used for verification. Qian and Chen [105] presented a model checking based testing technique for Web service composition. They used guarded automata for specifying the service composition (written in WS-CDL) and SPIN for verifying the interactions of services.

Further, as an improvement over the previous ones, recent model-checking based verification techniques [14, 15, 106] support automation to a great extent. Bentahar et al. [14] proposed a modeling and verification technique for composite Web services. Their modeling aspect was based on separation of concerns between operational and control behaviors (interactions among Web services) of Web services. Their verification technique was model checking-based where they automatically generated Kripke model out of the given operational behavior.

Studies	Verification goal	WS Technology	Model checker
Foster et al. [91]	Workflow trace equivalence verification	BPEL	–
Fu et al. [92]	Interaction among Web services	BPEL	SPIN
Kazhamiakin et al. [93]	Interaction among Web services	BPEL	SPIN
Walton et al. [94]	Mult-agent Web services	–	SPIN
Fu et al. [95]	Realizability of conversation protocol	–	–
Pistore et al. [96]	Verification of business process model	BPEL	NuSMV
Cao et al. [97]	Verification of UML model for BPEL	BPEL	SPIN
Deutsch et al. [107]	Communicating data-driven Web services	–	–
Zheng et al. [99]	BPEL test cases	BPEL, WSDL	SPIN, NuSMV
Deutsch et al. [100]	Data-driven Web applications	WDL	–
Gu et al. [101]	Web service conversations in WSCL	WSCL	–
Parizek and Adamek [102]	Session-oriented interactions	BPEL	Java PathFinder
Shegalov and Weikum [103]	Correctness of recovery framework	–	–
Wan et al. [104]	Web service communities	–	MCMAS
Qian and Chen [105]	Test case specifications	WS-CDL	SPIN
Bentahar et al. [14]	Web service operational behavior	–	–
Sheng et al. [106]	Web service operational behavior	–	NuSMV
El Kholy et al. [15]	Mult-agent based Web service compositions	–	MCMAS
El Kholy et al. [108]	Multi-agent based Web service compositions	–	MCMAS
Ghannoudi and Chainbi [109]	Interaction among Web services	WSDL	UPPAL

Table 2.4: Summary of Different Studies on Web Services Interaction Verification Using Model Checking

Similarly, Sheng et al. [106] proposed an automated service verification approach based on the operational and control behavior. The coordination of operational and control behaviors at run-time was facilitated by conversational messages and their proposed automated verification technique was based on symbolic model checking. El Kholy et al. [15] presented a framework to capture and verify the interactions among multi-agent based Web services. In order to capture the interactions, they proposed a specification language that uses commitment modalities in the form of contractual obligations. Table 2.4 presents a summary of different studies on Web services interaction verification using model checking.

2.2.4 Web Services Compliance and Equivalence Verification

Dynamic reconfiguration of services in a composition scenario leads to the quest of whether the available services can substitute an unavailable service. The problem of service substitution is categorized as context-independent and context-dependent [110, 111]. Finding a context-independent substitute for a service is easy as it is computed directly on the basis of the WSDL documentations of services [112], whereas the process of finding a context-dependent substitute is complicated and requires a more sophisticated procedure [113]. Since substitutability is not a stand-alone expression, it is always studied with other accompanying terminologies such as compatibility [110, 114], behavioral equivalence [115, 116], and compliance/conformance [117].

Pathak et al. [111] presented the concept of environment based substitutability notions: environment dependent and environment independent. Their logical formulation of the solution is based on the well-studied notion of “quotienting". Both and Zimmermann [118] analyzed Web service composability for compatibility, replaceability, and process conformance. Their approach enables one to verify if a newly added service holds the same rule as the previously existing one. For this purpose, they used the process algebra nets (PANs). Salah et al. [119] proposed a technique to check whether a concrete executable BPEL process conforms to its published interface behavior. In the process, they translated the concrete BPEL processes into PROMELA model and interface expressions into trace assertions. Gunay and

Yolum [120] presented an agent-based technique for semantic matchmaking (complete or partial) of services. They expressed the service requests as the properties written in LTL, and modeled the services using PROMELA.

Lomuscio et al. [121] presented an temporal-epistemic logic based technique for the verification of compliance/violations of predefined behaviors or contracts. The compatibility between two services ensures that they interact properly without any error. Guermouche and Godart [122] proposed a model checking based framework to deal with verifying the compatibility of a choreography where asynchronous communicating Web services are considered. Bersani et al. [123] proposed a SMT-based formal technique for runtime checking of Web service substitutability. Yeung [124] proposed a notion of choreography conformance in terms of Communicating Sequential Processes. Further, by using the proposed notion, he proposed an model checking based approach for service composition and verification.

Camera et al. [125] proposed an interactive tool-based approach for specification and verification of Web services behavioral adaptation contracts. Groefsema et al. [126] presented an approach for preventive compliance checking using temporal logics and model checking techniques. In order to get the interpretation model for verification, they proposed a mapping of service compositions to Kripke structures by using colored Petri nets. Huynh et al. [127] proposed a logic-based technique for finding similarity between Web services. Bataineh et al. [128] proposed a model checking based technique for verifying the compliance of autonomous and intelligent agent-based services with contracts regulating their composition specified in BPEL. Table 2.5 presents a summary of different studies on Web services compliance and equivalence verification using model checking.

Studies	Verification goal	WS Technology	Model checker
Pathak et al. [111]	Context specific substitutability of services	–	–
Both and Zimmermann [118]	Recursive and parallel BPEL systems	BPEL, WSDL	–
Salah et al. [119]	Conformance between behavioral interface and WS-BPEL	WS-BPEL	SPIN
Gunay and Yolum [120]	Web services matchmaking	OWL-S, SWRL	SPIN
Lomuscio et al. [121]	Compliance in agent-based service composition	OWL-S	MCMAS
Guermouche and Godart [122]	Choreography compatibility analysis	–	UPPAAL
Bersani et al. [123]	Runtime checking of service substitutability	–	–
Yeung [124]	Choreography conformance	WS-CDL, BPEL	–
Camara et al. [125]	Behavioral adaptation contract	WSDL, BPEL	MuCRL2
Groefsema et al. [126]	Compliance verification of Web service composition	–	NuXMV
Sbai et al. [129]	Compatibility	BPEL	NuSMV
Huynh et al. [127]	Web service similarity	–	OnTheFly WSCV
Bataineh et al. [128]	Compliance verification of Web service composition	BPEL	MCMAS

Table 2.5: Summary of Different Studies on Web Services Compliance and Equivalence Verification Using Model Checking

2.2.5 Web Services Non-Functional Requirements Verification

Non-functional requirements of Web services are mainly concerned with quality of service (QoS) and security related properties such as privacy, authorization, access control, etc. In order to detect the potential insecure information leakage, Nakajima [130] presented an idea of a lattice-based security label into BPEL. Nakajima, further, analyzed both the safety and security aspects in a single framework using the model-checking based verification techniques. Schlingloff et al. [50] presented an integrated technique for modeling and automated verification of Web service composition. Their modeling was based on Petri net and they employed model checking technique with alternating temporal logics for correctness verification. Mongiello and Castelluccia [131] presented a formal framework for modeling and verification of business processes written in BPEL. In this framework, modeling is done by translating a given BPEL document into the SMV language and verification is done automatically using NuSMV. In order to address the authorization requirements, Xiangpeng et al. [132] proposed a verifiable framework that integrates Role Based Access Control (RBAC) into BPEL. Further, with the help of model checking, they presented their verification technique.

In order to protect a session between Web services and a SOAP message by the derivation of keys, Xiaolie and Lejian [133] presented a secure session protocol using *WS-Trust* specification and *WS-SecureConversation* specification. Further, the security of the protocol was verified by using a security analysis tool AVISPA. To specify web services composition behavior and property, Zhang et al. [134] used a temporal logic based software architecture description language XYZ/ADL. Then, they translated the temporal logic description to timed-automata and applied refinement checking to verify the correctness of web service composition. Kasse and Mokdad [18] proposed a stochastic comparison based algorithm to check formulas with rewards on multidimensional continuous time Markov chains (CTMC) for composite Web services.

Zhao et al. [73] proposed an approach to generate Web service and BPEL test suit using model checking. The approach takes the advantage of model checking to verify BPEL script at the logical level, and to generate test suit automatically based on the model description, and finally to select test cases with respect to the counterexamples of model checking output. The approach contributes a set of algorithms and its implementation to support a translation from BPEL to LOTOS, and LTS to TTCN behavior tree. Gao et al. [135] studied the monitoring issue of Web service and proposed a predictive service monitoring approach to ensure availability of high-quality Web services at run-time. Considering functional and non-functional requirements, Gao et al. [135] used a probabilistic model checking technique to quantitatively verify the interactive behavior of Web services.

Lu et al. [136] proposed a model checking based approach to verify the satisfiability of behavior-aware privacy requirements in service composition. They extracted LTL specification from the behavior constraints of privacy requirements and they modeled the behavior of BPEL process by extending the interface automata, which supports privacy semantics. Then, it is transformed to PROMELA description and verified using SPIN. Jingjing et al. [137] proposed a timed automata based web service composition model. They provided a web service interface description language and a composition automation engine. Further, they validated its performance by using UPPAAL. Rebai et al. [19] proposed a formal verification approach using SPIN model-checker. The approach automatically transforms WS-CDL choreography specifications to PROMELA code and verifies non-functional properties that are written in LTL. Rui-Min and Xiao-Bin [138] proposed a Web service model diction algorithm of non-centralized automaton based on satisfiability modulo theories. In the proposed approach, SMT was used for bounded model checking of timed automaton, and the timed automaton model was directly converted into logical formulas identifiable by SMT.

Mi et al. [139] proposed a method to verify the reliability of BPEL processes. They extracted the model from a BPEL process and analyzed it through probabilistic model checking with PRISM. During the extension process, they added reliability attribute to each invoked sub-services. By structure extraction, the BPEL process was transformed to a PLTS system. Then, they generated a suitable analysis Markov model according to the feature of the PLTS model. Finally, they employed PCTL formula to describe the properties of the system, and verified it with PRISM.

Studies	Verification goal	WS Technology	Model checker
Nakajima [130]	Safety and security	BPEL	SPIN
Schlingloff et al. [50]	Usability, composability, and abstraction	BPEL	–
Mongiello and Castellucia [131]	Reliability of the business process design	BPEL	NuSMV
Xiangpeng et al. [132]	Role Based Access Control (RBAC)	BPEL	SAL
Xiaolie and Lejian [133]	Secure session protocol	–	AVISPA
Zhang et al. [134]	Refinement checking of Web service composition	–	UPPAAL
Kasse and Mokdad [18]	Quality of Service (QoS)	–	–
Zhao et al. [73]	Web service reliability verification	BPEL	–
Gao et al. [135]	Reliability prediction of Web services	–	–
Lu et al. [136]	Web service behavior-aware privacy requirements	–	–
Jingjing [137]	Feasibility of automatic web service composition	–	UPPAAL
Rebai et al. [19]	Non-functional properties	WS-CDL	SPIN
Rui-Min and Xiao-Bin [138]	Optimization	–	–
Mi et al. [139]	Reliability	BPEL	PRISM
ElKassmi and Jarir [20]	Privacy, integrity, authorization	–	UPPAAL
El-Qurna et al. [140]	Service trust behavior	–	MCFLC

Table 2.6: Summary of Different Studies on Non-Functional Requirements Verification of Web Services Using Model Checking

El Kassmi and Jarir [20] proposed a model checking based verification technique for Web service security requirements. This technique consists of modeling some security requirements using finite state machine (FSM), and their dynamic integration with expressed functional requirements. They used UPPAAL to validate the formalization and integration approach in web service composition. El-Qurna et al. [140]

proposed a model checking framework for service trust behaviors. They devised deterministic push down automaton (PDA) based trust behavior model. This model is built based on the observations' sequences, which are derived from the interactions with services. Further, they expressed the regular and non-regular trust behavior properties using fixed point logic with chop (FLC) and verified the properties using a symbolic FLC model checking algorithm. Table 2.6 presents a summary of different studies on non-functional requirements verification of Web services using model checking.

2.2.6 Web Services Time Requirements Verification

In this subsection, we review the model checking based Web service verification approaches that focus on the verification of time requirements in the context of Web services composition. Díaz et al. [141] presents an approach for analyzing and verifying the time requirements of Web service compositions via goal-driven models and model checking techniques. They employed a goal-driven model that was an extension of the goal model KAOS. The goal model specified the properties that the system must satisfy and the UPPAAL was used to verify the model. Kazhamiakin et al. [142] proposed a formal technique to capture and verify Web service compositions time related properties, where compositions are modeled using BPEL4WS. They defined a timed state transition system for Web services, namely WSTTS that can capture the timed behavior of web services in a composition scenario. Further, they employed interval temporal logic to specify the system's behavior (timed requirements) and used model checking algorithms for the verification purpose. Díaz et al. [98] presented an analysis technique for the timed behavior of Web services. Their technique is based on the translation of service description into the timed automata, and then, they used UPPAAL tool to simulate and analyze the system behavior.

Martinez et al. [143] proposed a technique for the design and verification of time requirements about Web services compositions by using the Web services translation tool. Using a timed automata representation, they demonstrated how to design a desired system and the verification of some properties on it. Mei et al. [66] proposed a formal technique to verify the time related properties of Web service compositions (modeled as BPEL4WS processes). They defined a timed automata (namely WS Timed Automata) for Web services that can capture the timed behavior of the web service compositions. Further, they proposed translation procedures to translate BPEL4WS descriptions into WS timed automata. The generated time automata was verified against the requirements using UPPAAL. Cambronero et al. [17] proposed a technique for the validation and verification of composite Web service systems with timing restrictions. They defined an operational semantics for a relevant subset of the WS-CDL. Further, they defined a translation of the considered subset of WS-CDL into a network of timed automata. They used the UPPAAL for the validation and verification of the described system, by using the generated timed automata.

Rawand et al. [144] presented a formal technique for modeling and verification of Web service compositions, where Web services were modeled by using open workflow nets (oWF nets). They verified the temporal requirements (written in CTL) against a Web service composition model (written as a SMV code) by using NuSMV. Table 2.7 presents a summary of different studies on Web services time requirements verification using model checking.

Studies	**Verification goal**	**WS Technology**	**Model checker**
Díaz et al. [141]	Time requirements	WS-CDL	UPPAAL
Kazhamiakin et al. [142]	Time requirements	BPEL	–
Díaz [98]	Times behavior of Web services	WSCI-WSCDL	UPPAAL
Martinez et al. [143]	Time restrictions in service composition	–	UPPAAL
Mei et al. [66]	Time properties	–	UPPAAL
Cambronero et al. [17]	Composite Web services with timing requirements	WS-CDL	UPPAAL
Rawand et al. [144]	Temporal properties	–	NuSMV
Zhao et al. [16]	Timed behavior of choreography	–	PAT

Table 2.7: Summary of Different Studies on Web Services Time Requirements Verification Using Model Checking

2.2.7 Multiple Verification Goals

In this subsection, we review the model checking based Web service verification approaches that focus on the verification goals belonging to more than one classes. Nakajima [145] identified faulty flow descriptions (written in WSFL) by the use of software model-checking technology. The properties checked were reachability, deadlock-freedom, or application specific progress properties. LTL was used for expressing the application specific properties. Betin-Can et al. [146] presented a verifiable design pattern and based on the pattern, they presented a modular verification approach to check the global behavior of services, their interfaces, and conversation among the services. Xiangpeng et al. [147] proposed a language CDL as a formal model of the simplified WS-CDL. Further, they translated WS-CDL to the input language of the model checker SPIN, which verifies the correctness of a given choreography. Dìaz et al. [148] presented a technique for the correct generation of WS-BPEL skeleton documents from the WS-CDL documents by using the timed automata and UPPAAL model checker. Vaz and Ferreira [149] proposed the use of model checking for the verification of Web service business process behavior and workflow patterns by translating the patterns into PROMELA.

Hongli et al. [150] proposed a choreography language CDL by which they verified control-flow properties and channel-passing related problems. Quyet et al. [151] presented a business process verifying technique to translate BPEL processes into PROMELA programs by using the labeled control flow graphs (LCFGs). Gao et al. [152] proposed a method for modeling the behavior of atomic services by using the interface automata and verifying the requirement specifications in a quantitative way. Ibrahim and Khalil [153] defined a set of transformation rules to generate a model that can be verified using the model checker UPPAAL. Huynh et al. [154] proposed a collection of approaches to address the problem of state-space exploration in the process of model checking of Web services. Their proposed approaches include a LTS-based model, heuristics strategies to find the best potential composition, and a bitwise indexing mechanism for fast location of suitable Web services. Table 2.8 presents a summary of different studies on Web services model checking having multiple verification goals.

2.2.8 Unclassified Verification Goals

In this subsection, we review the model checking based Web service verification approaches whose verification goals do not belong to any of previously mentioned verification classes. Lomuscio et al. [155] proposed a multi-agent based approach to model check Web service composition. Along with Web services safety and liveness properties, they also considered epistemic properties for the analysis and verification purpose. Their approach was able to capture the epistemic (knowledge of processes) modalities in addition to the temporal modalities and was also able to reason about these modalities. Wieczorek et al. [156] proposed model-based integration testing (MBIT) methods for the service choreographies called message choreography models (MCM). Further, MCMs were translated into Event-B models and provided as input to the test suit generator which used the model checker ProB. Peng et al. [157] proposed a model checking based verification approach for the channel passing related problems in choreography. They defined a WS-CDL based choreography language, namely $Chor_c$, in which basic service interaction patterns are represented by using pre and post conditions. Further, for the verification purpose, a $Chor_c$ description is translated into a model acceptable by the SPIN model checker.

Kang and Wang [158] extended WS-CDL with an XML-based new formal language, namely TLA4CDL, that is based on the concept of temporal logic of actions. The prime advantage of TLA4CDL is that it can capture action-action and action-state relationship which is not possible with only temporal modalities enabled logics such as LTL. Further, they proposed a model checking algorithm that was based on the concept of TLA4CDL.

Studies	Verification goal	WS Technology	Model checker
Nakajima [145]	Web service flow description and interaction verification	WSFL	SPIN
Betin-Can et al. [146]	Web service behavior, interface, and conversation verification	BPEL	SPIN
Xiangpeng et al. [147]	Semantics of WS-CDL	WS-CDL	SPIN
Díaz et al. [148]	Correctness of the services, time constraints, interaction and deadlock	WS-CDL, BPEL	UPPAAL
Vaz and Ferreira [149]	Workflow and business process behavior	–	SPIN
Hongli et al. [150]	Control-flow properties and channel-passing related problems	–	SPIN
Quyet et al. [151]	Business process control-flow and service interaction	BPEL	SPIN
Gao et al. [152]	Atomic service behavior and performance	–	–
Ibrahim and Khalil [153]	Control-flow and non-functional properties	BPEL	UPPAAL
Huynh et al. [154]	Functional properties and QoSs	–	PAT

Table 2.8: Summary of Different Studies on Web Services Model Checking Having Multiple Verification Goals

Fang et al. [159] presented an approach to automatically discover the suitable services and compose them by using the concept of bounded model checking. Particularly, given a set of available Web services and a requirement specification (written in LTL), the approach discovers the services that satisfies the requirement specification. In case, if no such suitable service is available, it composes the available services to realize the specification by keeping the bound (say K) in mind. Rossi [160] proposed a model checking and logic based algorithm for the verification of service compositions. The algorithm was filter (that prevents service misbehavior) based and intended for multileveled service compositions. Her focus was to verify security and correctness requirements written as security policies that are dynamically defined by the service participants or users. Oghabi et al. [86] proposed a verification approach for the individual Web services using OWL-S, where probabilities are introduced to model uncertainties about choice. Here, a Web service description was considered as input that was analyzed and translated into a discrete time Markov chain (DTMC) or Markov decision process (MDP) automatically. Then, the DTMC/MDP, resulted from the translation, was processed and modeled using the PRISM language. Further, by using the PRISM model checker, this model was verified to identify the properties that were being satisfied by the considered Web services.

Lomuscio et al. [161] proposed a verification technique for contract-regulated service compositions. They specified Web services and the governing contracts as WS-BPEL behaviors. Further, in order to verify the compliance or violation of contracts they employed an extended version of temporal-epistemic logic. The advantage with their approach was that they made contracts verifiable by representing it as WS-BPEL behaviors instead of writing it in a natural language. Yongxiang et al. [162] proposed a modeling and verification approach for the composition of cloud manufacturing services. Their proposed technique was based on unified modeling language (UML) and a process algebra based Web service orchestration calculus, namely COWS. They used COWS to formally specify the behaviors of services and they employed UML to compose cloud manufacturing services. A service composition modeled as activity diagram is converted into COWS syntax and considered as an input for the verification process. Kim et al. [163] proposed a formal approach to verify correctness claims and conflicts that are possible while a composite service is executing. Based on the application time, they classified the correctness requirements into four classes: assumptions, policies, pre-conditions, and post-conditions. OWL-S and SWRL were used to model composite services and the requirements respectively. Further, they provided automatic conversion methods to convert OWL-S and SWRL into PROMELA and LTL respectively, later, which are verified using the model checker SPIN.

Kokash and Arbab [164] proposed a formal framework for modeling and verification of long-running transactions (LRTs). They modeled and verified the LRTs by using a channel-based coordination language, namely Reo. By assuming that participant entities do not know the existence of each other in advance, with the help of Reo, a designer can analyze and verify the transactional behaviors of the system before the system's actual implementation. Bourne et al. [165] proposed a modeling and verification technique that captures and verifies the transactional properties (component and composition level) of composite Web services. Here, a user can specify the transactional requirements using temporal logic constructs, later, their conformance is verified using model checking based techniques. Table 2.9 presents a summary of different studies on Web services requirements verification using model checking where goals are not classified.

2.3 Analysis of SLR Results

Studies	**Verification goal**	**WS Technology**	**Model checker**
Lomuscio et al. [155]	Temporal and epistemic properties	–	MCMAS
Wieczorek et al. [156]	Integration testing	–	ProB
Peng et al. [157]	Channel passing in choreogrpay	WS-CDL,BPEL	–
Kang and Wang [158]	Temporal and action attributes	WS-CDL	–
Fang et al. [159]	Service discovery	–	–
Rossi [160]	Non-interference property, deadlock, livelock	–	–
Oghabi et al. [86]	Individual Web service design	OWL-S	PRISM
Lomuscio et al. [161]	Contract regulated Web service composition	BPEL	MCMAS
Yongxiang et al. [162]	Cloud manufacturing services	–	CMC
Kim et al. [163]	Conflict detection during the execution time of composite services	OWL-S, SWRL	SPIN
Kokash and Arbab [164]	Long-running transactions	WSDL, BPEL	mCRL2
Bourne et al. [165]	Transactional requirements of Web service compositions	–	NuSMV

Table 2.9: Summary of Different Studies on Web Services Requirements Verification Using Model Checking Where Goals are Not Classified

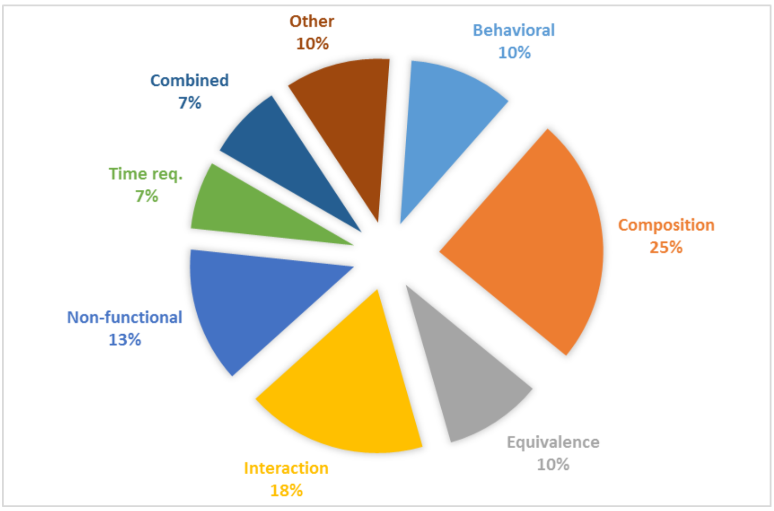


Figure 2.3: Percentage Distribution of the Articles Based on Their Verification Goals

Analysis of this SLR on model checking of Web services is made addressing the research questions discussed in Table 2.1. Fig. 2.3 shows the percentage distribution of the 116 articles, collected by us, based on the classification of their verification goals. Fig. 2.4 presents the year-wise distribution of the articles based on their verification goals. By analyzing Fig. 2.3 and Fig. 2.4, we deduce that most of the previous works are done in the areas of composition and interaction verification, and these topics are still receiving considerable attention from the research communities. One possible reason behind this fact could be paucity of the standard (widely accepted) model checking based approaches for the Web service verification.

Fig. 2.5 shows the percentage distribution of the selected articles based on the Web service standards they focus on in their proposed verification approaches. The highest number of the articles (34%) do not focus on any particular Web service standard. Here, based on the definition of basic and composite Web services, the researchers proposed their own modeling and verification techniques. Further, we see that BPEL is the most used Web service standard as it gives rise to several verifiable issues like concurrency, control-flow, etc. From Fig. 2.5, one can easily deduce that WSDL is not as much targeted as supposed to be targeted. One of the possible reasons behind this fact could be that the information provided by a WSDL document is specific to a Web service and not sufficient to investigate all the possible interaction or composition patterns. Due to this fact, researchers proposed the idea of annotat-

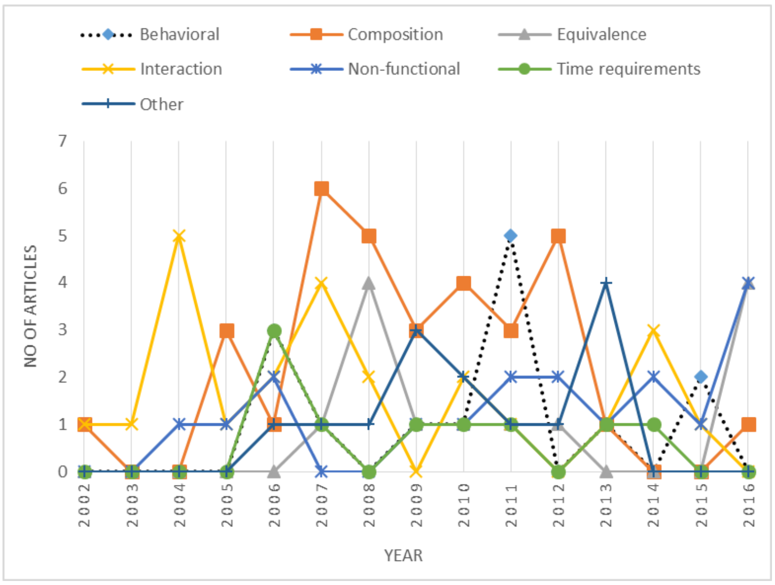


Figure 2.4: Year-Wise Distribution of the Articles Based on Their Verification Goals

ing the Web service standards [56, 61]. In earlier years, several Web service standards such as WSFL [166], WSCI [167], etc. were proposed and model checked. However, they have become obsolete and we do not find any recent verification works based on these standards.

Table 2.10 presents a summary of the popular model checking tools used for the Web service verification. In Table 2.10, each tool corresponds to a different flavor of the model checking and they differ from one another mainly in their verification goal. Fig. 2.6 presents the distribution of the articles based on their use of model checking tools. From Fig. 2.6, we can observe that 27% of our selected articles have not been used any model checking tools for their proposed model checking based Web service verification approaches. A possible reason behind this fact could be that the model checking technique is not applicable in the case of Web service verification as it is. In most of the cases, based on the primitive characteristics of Web services, some modifications has to be done in the classical model checking techniques. In case, if one does not modify the model checking concept then she has to take help of intermediate representations and/or other modeling and verification techniques. For instance, Table 2.11 shows that in many of the collected articles, other modeling and verification techniques are also used along with a model checking

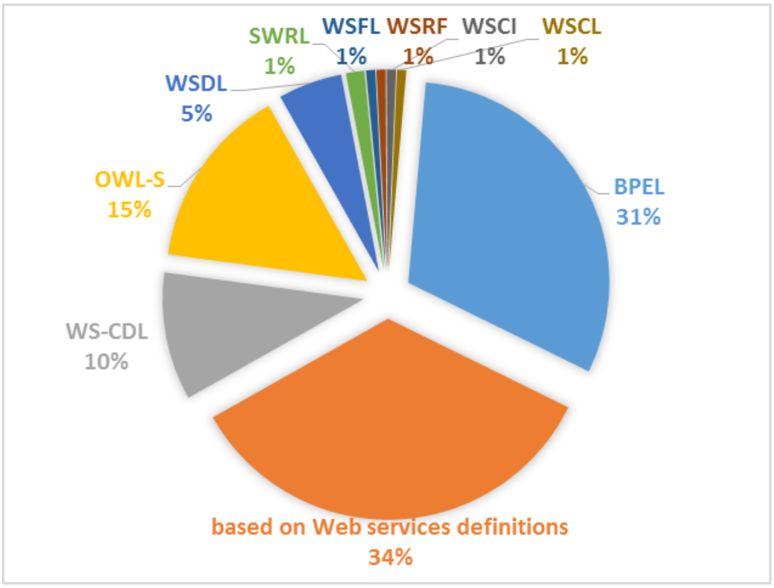


Figure 2.5: Percentage Distribution of the Articles Based on Their Focus Web service Standards

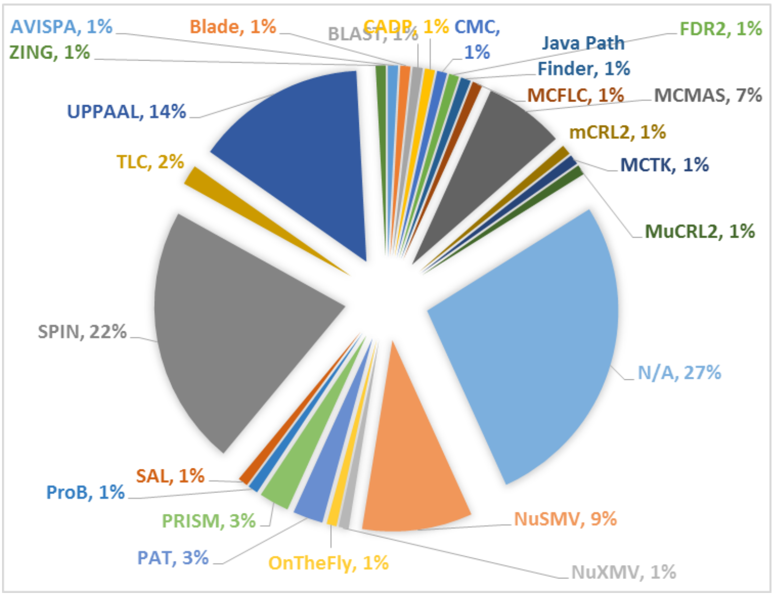


Figure 2.6: Distribution of the Articles Based on Their Use of Model Checking Tools

Tool Name	**Modeling Language**	**Specification Language**	**Application**
SPIN [168]	PROMELA	LTL	to trace logical design errors in the Web service compositions
NuSMV [169]	SMV	LTL, CTL	for the verification of synchronous and asynchronous communications among Web services
UPPAAL [170]	Timed automata	TCTL	for modeling, validation, and verification of real-time properties of Web services
PRISM [171]	PEPA	CSL, PLTL, PCTL	for formal modeling and analysis of random or probabilistic behavior of Web services
mCRL2 [172]	mCRL2	μ-calculus	for modeling, validation and verification of concurrency primitives in the context of Web services
PAT [173]	CSP	LTL	for composing, simulating and reasoning of concurrent and real-time systems
ZING [174]	C, Java	–	for exploring states of concurrent software systems
BLAST [175]	C	Monitor automata	to check that Web services satisfy the behavioral properties of the interfaces it uses
MCMAS [176]	ISPL	CTL, CTLK	for the verification of Web services modeled as Multi-Agent Systems (MAS)

Table 2.10: Summary of Different Model Checking Tools Used in the Web Services Verification Processes

technique. Moreover, we observe that in earlier years, model checking alone was used for verification, whereas in recent years focus is shifted towards the hybrid approaches to achieve more expressive and reasoning power.

Threats to Completeness. In order to conduct a systematic literature review, a reviewer construct a search string in such a way that she could collect all relevant studies without spending much effort and time. For this SLR, we searched seven databases and constructed the search string in such a way that it can include all the articles that are anyhow related to model checking and Web services. Further, we refined our initial selection by using the inclusion and exclusion criteria mentioned in Section 2.1. The search string for initial selection was flexible enough to allow

Key Studies	Supplementary Technique	Verification Goal Classification
[16, 64, 75, 80]	CSP	Behavioral, concurrency, composition, time requirements
[73, 77]	LOTOS	Composition, Non-functional
[118][63]	π-calculus	Equivalence, composition
[50]	Petri net	Non-functional
[126]	CPN	Equivalence
[61]	TCPN	Composition
[12, 81, 158]	TLA	Composition, behavioral
[97, 104, 124, 162]	UML	Interaction, equivalence
[87, 129, 144]	oWF	Time requirements, behavioral, equivalence
[151]	LCFG	Combined
[130]	Lattice	Non-functional
[83, 123, 138]	SMT	Equivalence, behavioral, Non-functional
[11]	Event calculus	Composition
[74]	PDDL	Concurrency

Table 2.11: Other Techniques Used with Model Checking

all the relevant articles, however, some articles might be missing due to linguistic barriers.

Threats to the Selection of Studies. Since different researchers may have different views on final selection of the studies, two researchers worked together to avoid the possibilities of any conflicts. However, in case of disagreement, other researchers from the same area were called to make a final decision on the selection of primary studies.

Threats to the Data Extraction. We searched seven databases and collected 582 articles in our initial selection. Thereafter, 116 articles were finally selected. We validated our search results by cross-checking individual journals and proceedings respectively.

2.4 Justification for the Present Work

In this chapter, we described an SLR on the referred research articles appeared during the period of 2002-2017 that are related to the model checking of Web services. Based on the verification goals, we classified the existing approaches into six classes: composition verification, interaction verification, behavioral verification, equivalence verification, non-functional requirement verification, and time requirement verification. Based on our review, we observe that (i) the area of Web service verification is still not matured. Researchers continue to propose new techniques for Web services verification, (ii) the model checking technique alone is not sufficient to model and verify all the primitive characteristics of Web services, (iii) Web service standards should be more enriched to support in-depth verification, and (iv) an integrated verification tool should be there to support multiple Web service standards and verification aspects. Based on our observations, in this thesis, we propose : (i) a formal modeling technique, namely *Recursive Composition Algebra (RCA)* to capture the notion of recursive composition among Web services, and (ii) RCA based verification techniques for verifying interactive and structural properties of the services. Further, we provide the implementation details of the proposed framework and as a cross-domain application, we perform the modeling of multistage attack using RCA.

Chapter 3

Recursive Composition Algebra (RCA)

In this chapter, we describe a *composition algebra* called Recursive Composition Algebra (RCA) for Web services based on the notion of recursive composition. Application of this algebra on Web services results in a graph, called recursive composition graph (RCG), that works as a model for the interpretation of the semantics of a given requirement specification.

3.1 RCA & RCG

3.1.1 Operands (Web Services) and Operators (Successor, Composition, and Recursive Composition)

Let $\mathcal{W} = \{w_1, \cdots, w_m, \epsilon\}$ be a finite set of available Web services, where ϵ represents an empty Web service. An empty Web service does not invoke any service or perform any activity. Throughout the thesis, we consider $\mathcal{W}$ in the same meaning unless stated otherwise. For the sake of convenience, Table 3.1 lists all the notations that are used in the proposed algebra. We define a Web service $w_i \in \mathcal{W}$ as follows:

Definition 3.1.1 (Web service)**.** A Web service $w_i \in \mathcal{W}$ is a 3-tuple $\langle I, O, Rl \rangle$, where $I = \{I_1, \cdots I_p\},\ p \in \mathbb{N}$ is a finite set of input messages, that w_i accepts. $O = \{O_1, \cdots, O_q\},\ q \in \mathbb{N}$ is a finite set of output messages, that w_i produces. Rl is a

Notation	Meaning
$\mathcal{W}$	a set of Web services
w	a Web service w
$w.I$	input set of Web service w
$w.O$	output set of Web service w
$w.Rl$	relation that maps an element of the input set of w to the power set of the output set of w
$\succ$	successor operator
$\succ_C$	conditional successor operator
$\succ_R$	restrictive successor operator
$\oplus$	composition operator
$\circledast$	recursive composition operator
$\langle w, I_p \rangle$	a service-input tuple such that $I_p \in w.I$
$w_i \oplus w_j \oplus \cdots \oplus w_n$	a composition chain
$\langle w_i \oplus w_j \oplus \cdots \oplus w_n, I_p \rangle$	a service input tuple such that $I_p \in w_n.I$

Table 3.1: List of Notations (with Their Meaning) Used in RCA

relation that maps an input message from I to output messages in O ($Rl \subseteq I \times 2^O$). $w_i.I$, $w_i.O$, and $w_i.Rl$ are referred as the set of input messages, the set of output messages, and the relation from $w_i.I$ to $2^{w_i.O}$ in w_i.

For a Web service, the set of input messages, the set of output messages, and the relation from input message set to output message set are static and available in the respective *Web Service Description Language (WSDL)* file.

Definition 3.1.2 (Absolute successor)**.** Let '$\succ$' be a symbol to represent the absolute successor operator. $\succ$ maps an element of the $\mathcal{W}$ to an element of the power set of the set $\mathcal{W}$ ($\succ$: $\mathcal{W} \rightarrow 2^{\mathcal{W}}$). Given a Web service $w_i \in \mathcal{W}$, $S \subset \mathcal{W}$ is the absolute successor of w_i if and only if $\forall w_j \in S \ : \ w_i.O \cap w_j.I \neq \emptyset$.

In other words, an absolute successor operator is an unary operator that provides services directly invokable by a service (we call them as successor services). Consider that a service $w_i \in \mathcal{W}$ invokes a service $w_j \in \mathcal{W}$. Then $w_j \in \ (\succ w_i)$. If w_j is not known in advance, we write $\succ w_i = w_{i+1}$ unless stated otherwise. If the service w_i directly invokes a set of services $(w_1, \cdots, w_l) \subset \mathcal{W}$ then $\succ w_i = \{w_1, \cdots, w_l\}$. If the service w_i does not invoke any service from the set $\mathcal{W}$ then $\succ w_i = \epsilon$.
Composition of services (say $n \in \mathbb{N}$, no. of services) is the aggregation of facilities

provided by the n services as a single service. Composability of a service with another service is decided by successor relation. Given a composite service w_i and its successor service w_j, w_j is always composable with w_i.

Let '$\oplus$' be a symbol that represents the composition. We define *service composition* as follows:

Definition 3.1.3 (Service composition)**.** Given two Web services $w_i, w_j \in \mathcal{W} : w_j \in (\succ w_i)$, composition of w_i and w_j (represented as $w_i \oplus w_j$) yields a composite Web service $w_k \in \mathcal{W}$ such that

$$\Big(\exists m \in w_i.I, \exists n \in w_j.I \ : \ w_i.Rl(m) = n\Big) \rightarrow \Big((m \in w_k.I) \ \wedge \ (w_k.Rl(m) \subseteq w_j.Rl(n))\Big)$$

Definition 3.1.4 (Service-input tuple)**.** A tuple $\langle w_i, I_p \rangle$ is in service-input tuple format if and only if $w_i \in \mathcal{W}$ and $I_p \in w_i.I$.

Definition 3.1.5 (Conditional successor)**.** A conditional successor ($\succ_C$) accepts the input and produces the output in the form of service-input tuple. Given a tuple $\langle w_i, I_p \rangle$, $\langle w_j, I_q \rangle$ is a conditional successor of $\langle w_i, I_p \rangle$ $\Big($written as: $\langle w_j, I_q \rangle \in \Big(\succ_C \langle w_i, I_p \rangle\Big) \Big)$ if and only if $w_j \in \Big(\succ w_i\Big)$ and $I_q \in w_i.Rl(I_p)$.

Let $\langle w_i, I_p \rangle$ and $\langle w_j, I_q \rangle$ be two service-input tuples such that $I_p \in w_i.I$, $I_q \in w_j.I$, and $w_i.Rl(I_p) = I_q$. Then, '$\langle w_i, I_p \rangle \oplus \langle w_j, I_q \rangle$' represents a composition chain that could participate in further composition processes as a single service. However, only the end elements of a composition chain participates in further composition process. If a service-message tuple $\langle w_i, I_p \rangle$ composes with $\langle w_j, I_q \rangle$ and $\langle w_r, I_r \rangle$ in parallel, it is represented by two separate composition chains: $\langle w_i, I_p \rangle \oplus \langle w_j, I_q \rangle$ and $\langle w_i, I_p \rangle \oplus \langle w_k, I_r \rangle$. A composition chain grows further with the attachment of other composable service-input tuples. However, a composition with an empty service results as a tuple itself without any change ($(\langle w_i, I_p \rangle \oplus \epsilon = \langle w_i, I_p \rangle)$). The conditional successor for a tuple with an empty second field (input message is not specified) behaves as an absolute successor, indicating that the service-input tuple can be replaced with the service name only. A conditional successor operator is a special case of restrictive successor operator ($\succ_R$) representing that $Domain(\succ_R$

$) = Domain(\succ)$ and $Range(\succ_R) \subseteq Range(\succ_C)$. We define a restrictive successor operator as follows.

Definition 3.1.6 (Restrictive successor)**.** Let $\langle w_i, I_p\rangle \oplus \langle w_j, I_q\rangle \oplus \cdots \oplus \langle w_n, I_s\rangle$ be a composition chain and $\langle w_x, I_r\rangle$ be a service-input tuple, then $\langle w_x, I_r\rangle \in \Big(\succ_R \langle w_i, I_p\rangle \oplus \langle w_j, I_q\rangle \oplus \cdots \oplus \langle w_n, I_s\rangle \Big)$ if and only if $\langle w_x, I_r\rangle \in \Big(\succ_C \langle w_i, I_p\rangle \oplus \langle w_j, I_q\rangle \oplus \cdots \oplus \langle w_n, I_s\rangle \Big)$ and $\langle w_x, I_r\rangle \notin \{\langle w_i, I_p\rangle, \langle w_j, I_q\rangle, \cdots, \langle w_n, I_s\rangle\}$.

The empty first field or second field in the input argument of a restrictive successor is a special case and is treated as follows:

$$\succ_R \langle w_i, -\rangle \equiv \succ_R \{\langle w_i, I_1\rangle, \langle w_i, I_2\rangle, \cdots, \langle w_i, I_p\rangle\} \tag{3.1}$$

where $\{I_1, I_2, \cdots, I_p\} = w_i.I$

$$\succ_R \langle -, I_p\rangle \equiv \succ_R \{\langle w_i, I_p\rangle, \langle w_j, I_p\rangle, \cdots, \langle w_l, I_p\rangle\} \tag{3.2}$$

where $\{w_i, w_j, \cdots, w_l\} \in \mathcal{W}$ such that $I_p \in w_i.I, w_j.I, \cdots, w_l.I$

Let '$\circledast$' be a symbol to represent recursive composition. To define recursive composition, we use restrictive successor operator ($\succ_R$) and composition operator ($\oplus$) as supplementary operators (defined earlier in this section).

Definition 3.1.7 (Recursive composition)**.** Recursive composition for a given service-input tuple $\langle w_i, I_p\rangle$ where $I_p \in w_i$ is defined as follows:

$$\circledast \langle w_i, I_p\rangle \triangleq \begin{cases} \epsilon & ; \text{if } \langle w_i, I_p\rangle = \epsilon \\ \langle w_i, I_p\rangle & ; \text{if } \succ_R \langle w_i, I_p\rangle = \epsilon \\ \langle w_i, I_p\rangle \oplus \Big\{ \circledast (\succ_R \langle w_i, I_p\rangle)\Big\}; & \text{otherwise} \end{cases} \tag{3.3}$$

Definition 3.1.7 infers that (i) if '$\circledast$' is applied over an empty service, then the result is the empty service, (ii) if '$\circledast$' is applied over a service-input tuple such that no restrictive successor is available for the tuple, then the result is empty service, and (iii) if '$\circledast$' is applied over a service-input tuple (other than the two above-mentioned tuples), then the result is a composition chain, where the input tuple is composed with '$\circledast$' applied over the restrictive successor of the input tuple.

Successor operator and recursive composition operator are having equal precedence. They possess higher precedence over the composition operator.

3.1.2 Algebraic Properties of the Operators $(\succ, \oplus, \circledast)$

The algebraic properties of proposed operators $(\succ, \oplus, \circledast)$ are given as follows:

- *Closure:* All three operators follow the closure property. It is assured that each operation produces services to which we can again apply operators.

- *Associativity:* Composition operator is associative. Successor and recursive composition are unary operators and possess right-to-left associativity.

$$(w_i \oplus w_j) \oplus w_k = w_i \oplus (w_j \oplus w_k)$$

- *Commutativity:* The composition operator is not commutative. As the remaining two operators (successor and recursive composition) are unary operators, commutativity is not defined for those operators.

- *Distributivity:* The composition operator is not distributive over other operators. Successor and recursive composition operators are distributive over a set of Web services. But they are not distributive over the composition operator.

$$\begin{aligned} \circledast(w_j \oplus w_k) &\neq (\circledast w_j) \oplus (\circledast w_k) \\ \succ (w_j \oplus w_k) &\neq (\succ w_j) \oplus (\succ w_k) \\ \circledast(w_i, w_j, \cdots, w_k) &= (\circledast w_i, \circledast w_j, \cdots, \circledast w_k) \\ \succ (w_i, w_j, \cdots, w_k) &= (\succ w_i, \succ w_j, \cdots, \succ w_k) \end{aligned}$$

  A set of Web services can be provided as an operand to the composition operator. The composition operation is distributive over a set of web services provided that every member service of the set is composable with the given operand. For instance, if $w_i \oplus w_j$ is a composite service and w_k and w_l are composable with w_j then

$$w_i \oplus w_j \oplus (w_k, w_l) = w_i \oplus w_j \oplus w_k, w_i \oplus w_j \oplus w_l$$

- *Identity:* An empty service (say, ϵ) is the identity element for the composition operation.

$$w_i \oplus \epsilon = w_i = \epsilon \oplus w_i$$

- *Inverse:* For a service w_i, the inverse service representation is $\overline{w_i}$.

$$w_i \oplus \overline{w_i} = \overline{w_i} \oplus w_i = \epsilon$$

The empty service (ϵ) is its own inverse.

3.1.3 Computability and Decidability of RCA

It is not always the case that an algebra or a mathematical function is computable. Therefore, it is inevitable to check the computability of our proposition. There are various computability models available [177] such as Turing-computability, μ-recursive functions, lambda calculus, etc. As per the structure of our proposition, we use the notions of Turing computability and μ-recursive functions to show that the RCA is computable.

Theorem 1. *Recursive composition operator* ($\circledast$) *is a primitive recursive function.*

Proof. A function is primitive recursive if it can be obtained from the initial functions by applying composition and primitive recursion, for a finite number of times. Successor, zero, and projection functions are the three well-known initial functions [178]. Here, composition stands for the standard definition of composition of functions [178] and the definition of primitive recursion states that if g and h are total number theoretic functions with n and $n+2$ variables respectively, then the $n+1$-variable function f defined by

1. $f(x_1, \cdots, x_n, 0) = g(x_1, \cdots, x_n)$
2. $f(x_1, \cdots, x_n, y+1) = h(x_1, \cdots, x_n, y, f(x_1, \cdots, x_n, y))$

is said to be obtained from primitive recursion. Here, the variable y is called as the recursive variable.
This definition of primitive recursion elucidates that

1. The function is recursively defined by using basic functions and composition among them.
2. There is a termination condition for the recursion.

3. Evolution of the recursion meets the termination condition.

Considering our proposed recursive composition operator, we show how all the three mentioned conditions are satisfied.

Condition 1. Let $\mathcal{W} = (w_1, \cdots, w_m, \cdots, \epsilon)$ be a finite set of Web services, where w_m is a terminator service (i.e. $\succ (w_m) = \epsilon$). By the definitions of *initial functions* [178], we deduce that $\succ_R$ is an initial function as it resembles completely with *successor function* and $\oplus$ is a composite function that comprises the basic functions. For the sake of convenience, let f represent the recursive composition operator, h represent the composition operator, and g represent the restrictive successor operator. The recursive composition operator (f) can be written in the terms of h and g as follows:

1. $f(w_i) = g(w_i) = w_i$ if w_i is an empty service
2. $f(w_i) = h(w_i, f(g(w_i)))$

Condition 2. Termination condition for this function is the occurrence of an empty service as an argument for the function f.

Condition 3. Termination condition becomes fulfilled only if $\succ_R (w_i) = \epsilon$; that is possible only if either w_i itself is an empty service or $\succ_R (w_i)$ is already being consumed as a part of composition chain in the composite service w_i. Evolution of recursion in recursive composition (defined using $\circledast$) is bound as the given set of Web services ($\mathcal{W}$) is finite. Further, a service in the composition chain that is already consumed during the formation of composition cannot be consumed again. The maximum possible length of a composition chain can be the cardinality of the $\mathcal{W}$. This assures fulfillment of the third condition. Hence, it is proved that recursive composition operator is primitive recursive. □

Theorem 2. *Recursive composition algebra* $(\mathcal{W}, \succ, \oplus, \circledast)$ *is Turing computable.*

Proof. A function is computable by a Turing machine if and only if it is a μ-recursive function. By the definition of μ-recursive function, which states that all primitive recursive functions are μ-recursive functions, all operators in recursive composition algebra are μ recursive functions. Since a function is computable by a Turing machine if and only if it is μ-recursive, the proposed algebra is Turing computable.

□

3.2 A Case Study on a Travel Agency Scenario

Let $\mathcal{W} = \{\, TA_1, TA_2, HB_1, HB_2, FB, CB, CT, EQ, CS, \epsilon \,\}$ be a finite set of Web services. TA_1, TA_2, HB_1, HB_2, FB, CB, CT, EQ, and CS are the abbreviations for services representing *travel agency1, travel agency2, hotel booking1, hotel booking2, flight booking, car booking, city tour, inquiry,* and *city weather report* respectively. ϵ represents an empty service that does not invoke any service or perform any activity.

For each Web service, there are two sets of messages: a set of input messages and a set of output messages. For the sake of easy understandability, in a composite service, the set of output messages consists of two subsets of messages. First one is set of response messages that have to be sent back as a reply to a requester and second subset consists of messages that have to be forwarded to other services. Input messages that end with question mark (?) are query messages (e.g., checking hotel availability). Rest of the input messages are service request messages (e.g., booking a hotel). Output messages that begin with negation symbol ($\neg$) are negative reply. In this example, two instances of travel agency services (TA_1 and TA_2) and hotel booking services (HB_1 and HB_2) are considered. However, input and output sets are not same in the case of travel agency services (TA_1 and TA_2). Following are the sets of input messages and output messages for the services mentioned in $\mathcal{W}$.

$$TA_1.I = \Big\{ Flight\_Avail?,\ F\_Book,\ Hotel\_Avail?, H\_Book,\ City\_Name \Big\}$$

$$\begin{aligned} TA_1.O = \Big\{ \Big(& Flight\_Yes, Flight\_No, F\_Bookd, \neg F\_Bookd, Hotel\_Yes, \\ & Hotel\_No,\ H\_Bookd,\ \neg H\_Bookd,\ Weather\_Report \Big) \\ & \Big(Flight\_Avail?,\ Flight\_Info?,\ Flight\_Enq?,\ F\_Book, \\ & Hotel\_Avail?,\ H\_Book,\ City\_Name \Big) \Big\} \end{aligned}$$

$$\begin{aligned}TA_2.I =&\Big\{Flight\_Info?,\ Hotel\_Avail?,\ Car\_Avail?,\ F\_Book,\ H\_Book,\\ &C\_Book\Big\}\\ TA_2.O =&\Big\{\Big(Flight\_Yes,\ Flight\_No,\ F\_Bookd,\ \neg F\_Bookd, Hotel\_Yes,\\ &Hotel\_No,\ H\_Bookd,\ \neg H\_Bookd,\ Car\_Yes,\ Car\_No,\\ &C\_Bookd,\ \neg C\_Bookd\Big)\Big(Flight\_Enq?,\ Hotel\_Avail?,\ Hotel\_Info?,\\ &Car\_Avail?,\ H\_Book, F\_Book,\ C\_Book,\ \Big)\Big\}\end{aligned}$$

$$\begin{aligned}HB_1.I =&\Big\{Hotel\_Avail?,\ H\_Book\Big\}\\ HB_1.O =&\Big\{\Big(Hotel\_Yes,\ Hotel\_No,\ H\_Bookd, \neg H\_Bookd\Big)\\ &\Big(Hotel\_Avail?,\ H\_Book\Big)\Big\}\end{aligned}$$

$$\begin{aligned}HB_2.I =&\Big\{Hotel\_Info?,\ H\_Rsrv\Big\}\\ HB_2.O =&\Big\{\Big(Hotel\_Yes,\ Hotel\_No,\ H\_Bookd, \neg H\_Bookd\Big)\\ &\Big(Hotel\_Avail?,\ H\_Book\Big)\Big\}\end{aligned}$$

$$\begin{aligned}FB.I =&\Big\{Flight\_Enq?,\ F\_Book\Big\}\\ FB.O =&\Big\{\Big(Flight\_Yes,\ Flight\_No,\ F\_Bookd, \neg F\_Bookd\Big)\\ &\Big(Flight\_Avail?,\ F\_Book\Big)\Big\}\end{aligned}$$

$$\begin{aligned}CB.I &= \Big\{Car\_Avail?,\ C\_Book\Big\}\\ CB.O &= \Big\{Car\_Yes,\ Car\_No,\ C\_Bookd,\ \neg C\_Bookd\Big\}\end{aligned}$$

$$\begin{aligned}CT.I &= \Big\{Tour\_Info?,\ T\_Book\Big\}\\ CT.O &= \Big\{Tour\_Plan,\ T\_Bookd,\ \neg Tour\_Bookd\Big\}\end{aligned}$$

$$EQ.I = \Big\{City\_Name,\ City\_Code\Big\}$$
$$EQ.O = \Big\{City\_Weather\_Report\Big\}$$

$$CS.I = \Big\{City\_Code\Big\}$$
$$CS.O = \Big\{City\_Weather\_Report\Big\}$$

In this example, we adopt a convention that for a service, an input message can be mapped to the output messages that begin with the same initials as of input message. For instance, let us consider an input message $Flight\_Avail?$ from the set $TA_1.I$. This input message can be mapped to the following output messages available in the set $TA_1.O$: $Flight\_No$, $Flight\_Yes$, $Flight\_Avail?$, $Flight\_Info?$, and $Flight\_Enq?$.

The illustrated travel agency scenario is considered for Web service composition, interaction, and equivalence verification in Chapter 4, 5, and 6 respectively.

3.3 Recursive Composition Graph (RCG) and Its Construction

Applying the recursive composition operation on a service (say w_i) generates a topologically sorted directed acyclic graph {called as Recursive Composition Graph (RCG)} with w_i as the root. We call every path (from the root to the leaf) in the graph as a *trace*. Let w_i be a Web service. $\mathcal{T}_{w_i}$ represents a set which contains all the traces generated by applying the recursive composition on w_i.

By using the travel agency scenario (described in Section 3.2), we illustrate the construction of a RCG as follows. Let us consider that a service-input tuple

$\langle TA_1, Hotel\_Avail?\rangle$ is provided as an input to the recursive composition operator.

$$\begin{aligned}
&\circledast\langle TA_1, Hotel\_Avail?\rangle = \langle TA_1, Hotel\_Avail?\rangle \oplus \Big\{ \circledast\, (\succ_R (\langle TA_1, Hotel\_Avail?\rangle))\Big\}\\
&=\langle TA_1, Hotel\_Avail?\rangle \oplus \Big\{ \circledast\, (\langle TA_2, Hotel\_Avail?\rangle, \langle HB_1, Hotel\_Avail?\rangle)\Big\}\\
&=\langle TA_1, Hotel\_Avail?\rangle \oplus \Big\{ \circledast\, (\langle TA_2, Hotel\_Avail?\rangle), \circledast(\langle HB_1, Hotel\_Avail?\rangle)\Big\}\\
&=\langle TA_1, Hotel\_Avail?\rangle \oplus \langle TA_2, Hotel\_Avail?\rangle) \oplus \Big\{ \circledast\, (\succ_R (\langle TA_2, Hotel\_Avail?\rangle))\Big\},\\
&\quad \langle TA_1, Hotel\_Avail?\rangle \oplus \langle HB_1, Hotel\_Avail?\rangle \oplus \Big\{ \circledast\, (\succ_R (\langle HB_1, Hotel\_Avail?\rangle))\Big\}\\
&=\langle TA_1, Hotel\_Avail?\rangle \oplus \langle TA_2, Hotel\_Avail?\rangle) \oplus \Big\{ \circledast\, (\langle HB_1, Hotel\_Avail?\rangle,\\
&\quad \langle HB_2, Hotel\_Info?\rangle)\Big\}, \langle TA_1, Hotel\_Avail?\rangle \oplus \langle HB_1, Hotel\_Avail?\rangle \Big\{ \circledast\, (\epsilon))\Big\}\\
&=\langle TA_1, Hotel\_Avail?\rangle \oplus \langle TA_2, Hotel\_Avail?\rangle) \oplus \Big\{ \circledast\, \langle HB_1, Hotel\_Avail?\rangle,\\
&\quad \circledast\, \langle HB_2, Hotel\_Info?\rangle\Big\}, \langle TA_1, Hotel\_Avail?\rangle \oplus \langle HB_1, Hotel\_Avail?\rangle\\
&=\langle TA_1, Hotel\_Avail?\rangle \oplus \langle TA_2, Hotel\_Avail?\rangle \oplus \Big\{\langle HB_1, Hotel\_Avail?\rangle\\
&\quad \oplus (\circledast(\succ_R (\langle HB_1, Hotel\_Avail?\rangle))), \langle HB_2, Hotel\_Info?\rangle \oplus (\circledast(\succ_R\\
&\quad \langle HB_2, Hotel\_Info?\rangle))\Big\}, \langle TA_1, Hotel\_Avail?\rangle \oplus \langle HB_1, Hotel\_Avail?\rangle\\
&=\langle TA_1, Hotel\_Avail?\rangle \oplus \langle TA_2, Hotel\_Avail?\rangle \oplus \Big\{\langle HB_1, Hotel\_Avail?\rangle \oplus (\circledast(\epsilon)),\\
&\quad \langle HB_2, Hotel\_Info?\rangle \oplus (\circledast(\epsilon)\Big\}, \langle TA_1, Hotel\_Avail?\rangle \oplus \langle HB_1, Hotel\_Avail?\rangle\\
&=\langle TA_1, Hotel\_Avail?\rangle \oplus \langle TA_2, Hotel\_Avail?\rangle \oplus \Big\{\langle HB_1, Hotel\_Avail?\rangle,\\
&\quad \langle HB_2, Hotel\_Info?\rangle\Big\}, \langle TA_1, Hotel\_Avail?\rangle \oplus \langle HB_1, Hotel\_Avail?\rangle\\
&=\langle TA_1, Hotel\_Avail?\rangle \oplus \langle TA_2, Hotel\_Avail?\rangle \oplus \langle HB_1, Hotel\_Avail?\rangle,\\
&\quad \langle TA_1, Hotel\_Avail?\rangle \oplus \langle TA_2, Hotel\_Avail?\rangle \oplus \langle HB_2, Hotel\_Info?\rangle,\\
&\quad \langle TA_1, Hotel\_Avail?\rangle \oplus \langle HB_1, Hotel\_Avail?\rangle
\end{aligned}$$

We get the following three traces based on the said derivation:
$\langle TA_1, Hotel\_Avail?\rangle \oplus \langle TA_2, Hotel\_Avail?\rangle \oplus \langle HB_1, Hotel\_Avail?\rangle$,
$\langle TA_1, Hotel\_Avail?\rangle \oplus \langle TA_2, Hotel\_Avail?\rangle \oplus \langle HB_2, Hotel\_Info?\rangle$,
$\langle TA_1, Hotel\_Avail?\rangle \oplus \langle HB_1, Hotel\_Avail?\rangle$.
Figure 3.1 depicts the RCG that is built based on these traces found out of the operation.

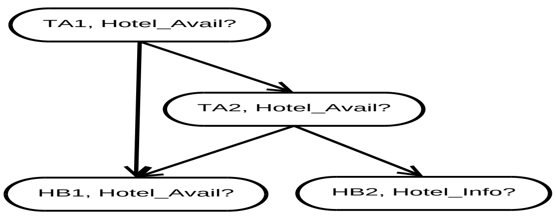


Figure 3.1: RCG Generated by Applying Recursive Composition on a Service-Input Tuple $\langle TA_1, Hotel\_Avail? \rangle$

Similarly, Figure 3.2 depicts the RCG generated by applying recursive composition on a service TA_1.

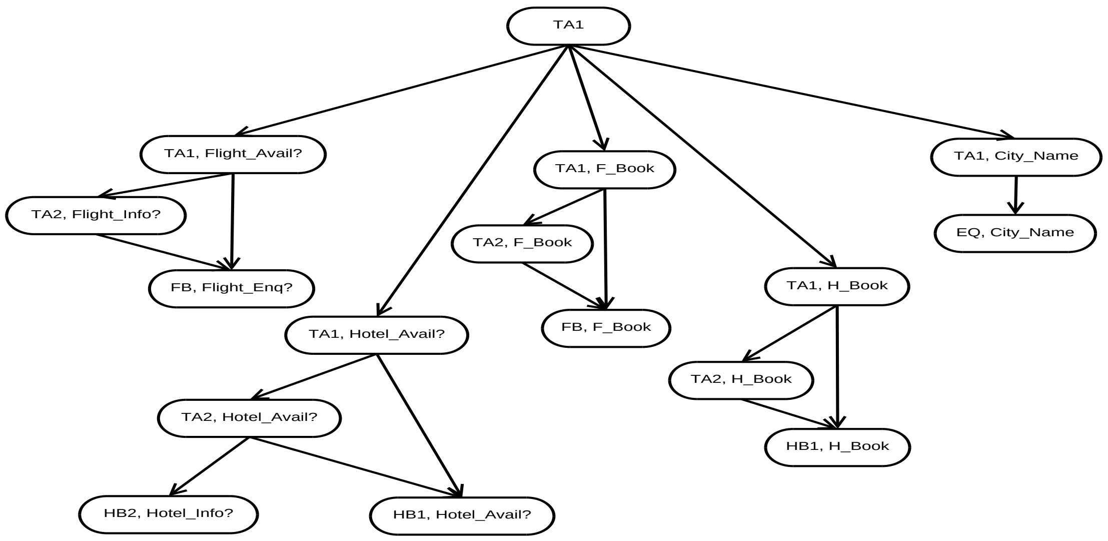


Figure 3.2: RCG Generated by Applying Recursive Composition on TA_1

A RCG does not allow repetition of a node in a trace, thus avoiding the deadlock. In a RCG, if a trace does not end with a basic service, then the trace is incomplete and this situation infers that the set of available services is not able to complete the particular demand. Therefore, we emphasize that every trace should end up with a node consisting a basic service.

3.4 Comparison of the Proposed Algebra with Other Existing Web Service Algebras

Several Web service algebras have been proposed for Web service composition modeling. In this section, we briefly overview and compare the approaches that are closely related to our work.

Hamadi and Benatallah [179] proposed a Web service algebra based on the Petri net model for the representation of services and interpretation of the algebraic syntax. Although this work [179] is considered as one of the novel approaches in Web service algebra, it is difficult to realize each member of a Web service set as a standalone Petri net model, considering the seamless proliferation of Web services nowadays. This prevents the algebra in [179] being practically implementable.

Hashemian and Mavaddat [180] presented a composition algebra as an alternative way of representing *interface automata*. This composition algebra captures the behavior of services and their composition. However, the authors explicitly mentioned that there is no direct or indirect recursion allowed in service composition [180]. Contrary to [180], our proposed algebra is focused on the recursive composition.

Hoefner and Lautenbacher [181] presented an algebraic structure of Web services which assisted users in Web service composition and formal description of their services. However, the syntax and semantics used in [181] do not seem intuitive for modeling and verification process.

Hu et al. [182] proposed a service net algebra (SNA) based on logic Petri net model. They emphasized the reuse of a service process instead of reusing a composite Web service. However, as mentioned by the authors, a drawback with [182] is that the Petri net is not suitable to model the complicated Web service composition.

Yu and Bouguettaya [183] proposed a Web service query algebra based on a formal service model that provided a high level abstraction of Web services across an application domain, aiming to describe a solution to service query. However, our proposed algebra aims to describe service composition and verification.

The aim and scope of process algebras are slightly different from the Web service algebra. However, several researchers applied process algebras in Web services [37,

Algebra	Operand	Operators	Underlying technique	Purpose	Modeling	Verification
Hamadi and Benatallah [179]	Web services as service nets	Sequence, alternative, iteration, discriminator, selection, and refinement	Petri net	To capture finer subtleties regarding Web services	Control flow and service combinations	-
Hashmian and Mavaddat [180]	Web services as input-output tuples	Sequence, choice, parallel, and synchronization	Process algebra	To provide easier alternative of interface automata	Web services as processes	-
Hofner and Lautenbacher [181]	Web services as Web methods	Composition, restriction, and iteration	Knowledge set based on WSDL and Web methods	To achieve goal-based automatic composition	Web service composition	-
Yu and Bouguettaya [183]	Web services as service graphs	F-map, Q-select, and compose	Service schema and service graphs	To query services efficiently and flawlessly	Querying the services	-
Hu et al. [182]	Web services as service nets	Sequence, parallel, alternative, and iteration	Labeled logic Petri net	To analyze substitution and projection	Web services substitution and proje.ction	-
Proposed Algebra	Web services as input-output tuples	Successor, composition, and recursive composition	Recursive composition algebra	To achieve automatic and dynamic composition	Recursive composition & service behavior	composition, interaction, equivalence

Table 3.2: Comparison of the Proposed Algebra with Other Existing Web Service Algebras

184, 185]. In general, process algebras are good in detecting behavioral equivalence or bisimulation study. Though process algebras treat the movement of a message across the chain of services efficiently, it differs from our proposed algebra in its treatment of mobility. A link between two processes is mobile in process algebras, whereas a link is stable in the case of modeling communication among Web services.

Table 3.2 presents a comparison of the proposed algebra with other existing Web service algebras.

3.5 Summary

In this chapter, we presented a recursive composition based algebra (namely RCA) for composition and verification of Web services. On applying over the set of Web services, the recursive composition operator yields a directed acyclic graph (RCG) that works as an interpretation model to verify the various kinds of requirement specifications about Web services. The rest of the thesis is built over RCA as the fundamental concept.

Chapter 4

Web Services Composition Verification Using RCA

In this chapter, we present a RCA-based formal technique for Web services composition verification at design level. This technique partitions a set of available Web services into disjoint subsets and makes a Web Services Set Partition (WSSP) graph. Further, it defines the concept of *goal set*, and deduces the logical inferences based on WSSP graph and goal sets.

4.1 Web Services Set Partition (WSSP)

Let $\mathcal{W} = \{w_1, \cdots, w_m\}$ be a finite set of Web services being considered for a composition scenario.

Definition 4.1.1 (Igniter Web service). A Web service $w_i \in \mathcal{W}$ is called as an igniter Web service if it initiates the chain of service composition.

Definition 4.1.2 (Isolated Web service). A Web service $w_i \in \mathcal{W}$ is called as an isolated service with respect to $\mathcal{W}$ if and only if $\succ w_i = \emptyset$ and $\nexists w_j \in \mathcal{W} : w_i \in (\succ w_j)$.

Given a set of Web services and an igniter service, Algorithm 4.1 partitions the set into several subsets. Algorithm 4.1 takes two inputs: a set of Web services $\mathcal{W}$ and an igniter service $w_i \in \mathcal{W}$, and it returns a set of subsets $\mathcal{S} = \{S_1, \cdots, S_n\}$ of $\mathcal{W}$, where $n < 2^{|\mathcal{W}|}$ and $\mathcal{S} \subset 2^{\mathcal{W}}$ such that the following four conditions hold

Algorithm 4.1 WSSPGRAPH-GENERATION($\mathcal{W}, w_i$)

```
    **Input:** $\mathcal{W} = \{w_1, w_2, w_3, \cdots, w_m\}$, $w_i \in \mathcal{W}$
    **Output:** $\mathcal{S} = \{S_1, S_2, S_3, \cdots, S_n\}$
 1: let $\mathcal{S} = \{S_1 \leftarrow \emptyset, \cdots, S_n \leftarrow \emptyset\},\ n = 2^{|\mathcal{W}|} - 1$ be a set
 2: $S_1 \leftarrow w_i$
 3: for all $w_i \in S_1$ do
 4:     $Temp \leftarrow \mathcal{W}$
 5:     $Temp \leftarrow \{Temp - S_1\}$
 6:     let $\mathcal{P} = \{P_1\}$ be a set
 7:     $P_1 \leftarrow S_1$
 8:     $n \leftarrow 1$
 9:     while $(P_n \neq \emptyset)$ do
10:         create a set $P_{n+1} \leftarrow \emptyset$
11:         $\mathcal{P} \leftarrow \mathcal{P} \cup P_{n+1}$
12:         for all $w_i \in P_n$ do
13:             for all $w_j \in Temp$ do
14:                 if $w_j \in (\succ w_j)$ then
15:                     $P_{n+1} \leftarrow w_j$
16:                 end if
17:             end for
18:         end for
19:         $Temp \leftarrow \{Temp - P_n\}$
20:         $S_n \leftarrow S_n \cup P_n$
21:         $n \leftarrow n + 1$
22:     end while
23: end for
24: for all $S_i \in \mathcal{S}$ do
25:     if $S_i = \emptyset$ then
26:         $\mathcal{S} \leftarrow \{\mathcal{S} - S_i\}$
27:     end if
28: end for
29: return $\mathcal{S}$
```

1. $\forall S_i \in \mathcal{S} : S_i \neq \emptyset$ (No partition set should be empty.)

2. $|S_1| = 1 \implies \Big(\forall S_i, S_j \in \mathcal{S} : S_i \neq S_j \implies S_i \cap S_j = \emptyset\Big)$. (Partition sets with respect to a single igniter are pairwise disjoint.)

3. $S_n \subseteq \bigcup\limits_{\forall w_i \in S_{n-1}} (\succ w_i)$, where $n > 1$. (Each successor partition set (except than the first set S_1) $S_n \in \mathcal{S}$ is subset or equal to union of service invocation possibility sets for each element of the predecessor partition set.)

4. $|\bigcup\limits_{i=1}^{n} S_i| < |\mathcal{W}| \iff \exists w_i \in \mathcal{W} : w_i \notin S_j, \text{where} 1 \leq j \leq n.$ (Non-exhaustive partition of $\mathcal{W}$ implies existence of the isolated services in the set.)

4.1.1 Web Services Set Partition Graph (WSSP Graph)

A Web Services Set Partition (WSSP) graph depicts all the subsets and restrictive successor services in a partitioned Web services set resulting from the application of Algorithm 4.1. In a WSSP graph, a $S_i \in \mathcal{S}$ behaves as a multi-element node, and a service invocation possibility is interpreted into directed edges from a Web service to another Web service. There are two types of directed edges used in this graph: dashed arrow and solid arrow. A dashed arrow indicates that the service on the arrow-head side is available on this composition chain but cannot directly be invoked by an arrow-tail side service. A solid arrow indicates that the service on the arrow-head side can be invoked directly by the service on the arrow-tail side. Fig. 4.1 is an example of WSSP graph. In Fig. 4.1, two order subscripts are used to name the services (for ease of representation). A WSSP graph can be constructed in two different ways: consolidated WSSP Graph and distinguished WSSP Graph.

Consolidated WSSP Graph. For a given set of Web services, a consolidated WSSP graph is formed by using Algorithm 4.1 with the condition that all strict igniters are placed in the first subset (S_1). Fig. 4.1 depicts a consolidated WSSP graph for a given set of Web services. All the services that can be invoked by any service in S_1 will be in the second subset (S_2) with their respective directed edges, and so on. Repetitive occurrences of a service in several subsets are possible. It is also possible that a service could invoke a service from its own set. Even though a service is invoking a service from its own set, an arrow must not be there within the subset.

Distinguished WSSP Graph. A distinguished WSSP graph is a consolidated WSSP graph with the condition that only one Web service can exist in the set S_1. Fig. 4.2 is a typical depiction of a distinguished view with w_{1j} as an igniter. A distinguished WSSP graph always forms a tree, where the igniter service placed in S_1 acts as the root node. In contrast to a distinguished WSSP graph, a consolidated WSSP graph is not restricted to form a tree.

A trace is a unidirectional tree $\langle V, E \rangle$ where vertices represent Web services, edges represent service invocation possibilities, and an arrow head shows the direction of workflow provided that each vertex is connected with exactly two edges (one is input and another is output) except first (root) and last (leaf) vertices. In other words, a trace is a linear Web services composition workflow path.

Trace Inclusion. Trace inclusion refers to the containment of a trace by another trace(s). Trace inclusion can be classified into total trace inclusion and partial trace inclusion. *Total trace inclusion* is a condition where the root of a trace (for example w_i) is present in a trace that is generated from another root (for example w_j). In this way, w_j contains all the traces generated by w_i. By computing total trace inclusion, we avoid the redundant traces in searching or performing some actions over traces. Given two or more traces generated by different igniters, a *partial trace inclusion* occurs if a trace-fraction is found common in all traces. For example, in Fig.4.1, a trace fraction from w_{rl} to w_{nl} is common in all traces generated by igniters Ig2, Ig3, and Ig4. By computing partial trace inclusion, the best-time complexity of a verification process could be improved. The occurrence of a trace inclusion is possible only in a consolidated view because of the existence of more than one igniter.

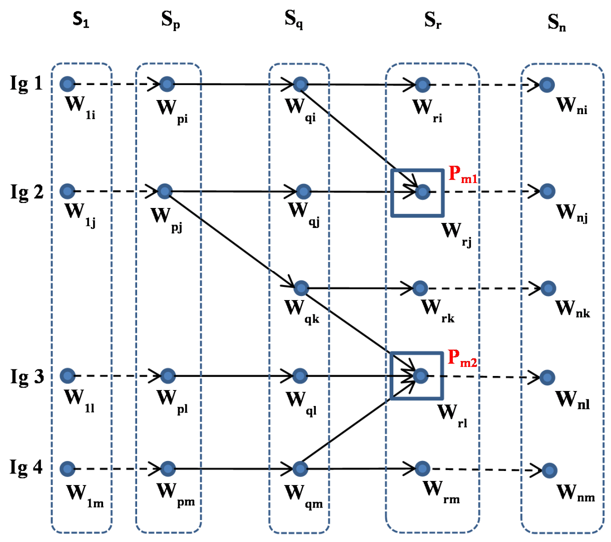


Figure 4.1: An Example of Consolidated WSSP Graph

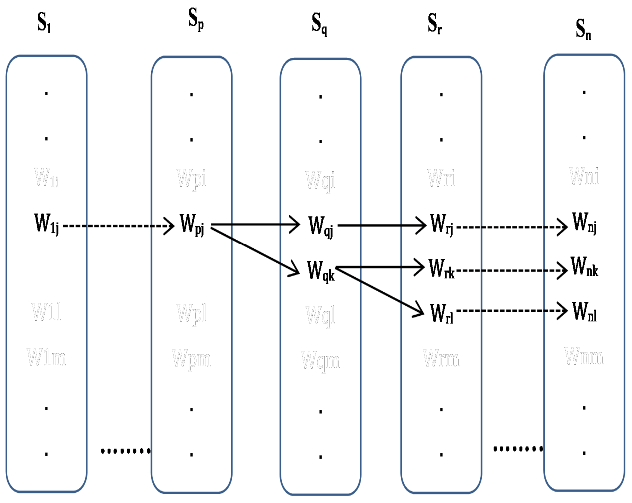


Figure 4.2: An Example of Distinguished WSSP Graph

Trace Merging. Trace merging is a condition where two or more traces originated from different igniters conjunct at a point. Trace merging can be classified into total trace merging and partial trace merging. For an igniter, a *total trace merging* situation occurs when all the traces originated from an igniter get merged at a point that lies on the trace generated from another igniter. In Fig. 4.1, for an igniter Ig3,

P_{m2} is a total merging point as the only trace originated from Ig3 is merged here. For an igniter, a *partial trace merging* situation occurs if some traces get merged but not all. In Fig. 4.1, for Ig1 and Ig2, P_{m1} is a merging point where each individual traces from both igniters conjunct. These are not the only traces from Ig1 and Ig2; besides these traces, there are some other traces which do not get merged. Merging points are very critical points and play an important role in the verification process.

4.2 Web Service Composition Verification

In this section, we present the concept of *goal set of a Web service*, and we present a verification technique for the possible deadlock conditions and behavioral equivalence between Web services using the goal sets of the Web services. The formal definition of the goal set of a Web service is given as follows.

Definition 4.2.1 (Goal set of a Web service)**.** Given a set of Web services $\mathcal{W}$, let $G(V, E)$ be a distinguished WSSP graph made by considering $w_i \in \mathcal{W}$ as the igniter service. Then, the goal set of w_i (written as $\mathcal{G}_i$) with respect to $\mathcal{W}$ consists of all terminal nodes in $G(V, E)$.

In other words, a subset S_n (resulted from the partitioning process) represents the goal set for a given WSSP graph. Algorithm 4.2 presents the procedure for constructing the goal set of a Web service that takes inputs $\mathcal{W}$ and $w_{ig}(\in \mathcal{W})$, and returns a goal set $\mathcal{G}_{ig}$. Let $\mathcal{G}_i$ be a goal set of a Web service w_i, and let '$\rightsquigarrow$' be a symbol that infers *leads to*. Then, '$w_i \rightsquigarrow \mathcal{G}_i$' represents the fact that w_i leads to the goal set $\mathcal{G}_i$. In case, if a service cannot invoke another services, its goal set consists of the service itself.

$$\text{If } w_i = \epsilon \text{ then } w_i \rightsquigarrow \mathcal{G}_i = w_i$$

4.2.1 Classification of a Given Set of Web Services

Let $\mathcal{W}$ be a given set of Web services, and let S_{nr}, S_{tm}, and S_{ig} be the subsets of $\mathcal{W}$ such that they represent non-reachable, terminator, and igniter sets of Web services respectively. Logical inferences deduced using the concept of goal sets and its parent sets are described as follows.

Algorithm 4.2 GOALSET-GENERATION($\mathcal{W}, w_{ig}$)

```
    Input: $\mathcal{W}, w_{ig}$
    Output: $\mathcal{G}_{ig}$
1:  $G(V, E) \leftarrow$WSSPGRAPH-GENERATION($\mathcal{W}, w_{ig}$)
2:  $\mathcal{G}_{ig} \leftarrow \emptyset$                    ▷ create an empty goal set $\mathcal{G}_{ig}$
3:  for each $v \in V$ do
4:      if $child[v] = \emptyset$ then              ▷ vertex $v$ is a terminal node
5:          $\mathcal{G}_{ig} \leftarrow key[v]$          ▷ $\mathcal{G}_{ig}$ is goal set of the service $w_{ig}$
6:      end if
7:  end for
8:  return $\mathcal{G}_{ig}$
```

A *non-reachable Web service* with respect to a given set of Web services is one that cannot invoke other services as well as no other service can invoke it. Exclusion of non-reachable services from the Web services set reduces the service discovery time during the process of automatic composition. By using the concept of WSSP graph and goal sets, non-reachable services could be automatically recognized by using the Expression 4.1.

$$(\exists w_i \in \mathcal{W})(\nexists w_j \in \mathcal{W})[(w_i \rightsquigarrow \mathcal{G}_i = \{w_i\}) \wedge (w_j \rightsquigarrow \mathcal{G}_j \neq \emptyset) \wedge (w_i \in P(\mathcal{G}_j))] \implies w_i \in S_{nr} \quad (4.1)$$

A *strict igniter service* is one that cannot be invoked by other services but can invoke other services. The strict igniter services can be identified by using the Expression 4.2.

$$(\exists w_i \in \mathcal{W})(\nexists w_j \in \mathcal{W})[(w_i \rightsquigarrow \mathcal{G}_i \neq \emptyset) \wedge (w_j \rightsquigarrow \mathcal{G}_j \neq \emptyset) \wedge (w_i \in P(\mathcal{G}_j))] \implies w_i \in S_{ig} \quad (4.2)$$

A *strict terminator service* does not invoke any service but other services can invoke it. Strict igniter services can be identified by using the Expression 4.3.

$$(\exists w_i \in \mathcal{W})(\exists w_j \in \mathcal{W})[(w_i \rightsquigarrow \mathcal{G}_i = \{w_i\}) \wedge (w_j \rightsquigarrow \mathcal{G}_j \neq \emptyset) \wedge (w_i \in P(\mathcal{G}_j))] \implies w_i \in S_{tm} \quad (4.3)$$

Classifying strict igniter and strict terminator services into different sets reduces the service discovery time while searching the suitable services for composition. The said classification of services is dependent on the given set of services. Any change in the set may bring the change in the classes resulted from the classification.

4.2.2 Goal Set Based Behavioral Equivalence Verification

In literature, we find that most of the service equivalence methods are merely based on the matching of the input-output sets of the services [186, 187, 188]. Determining service equivalence based on the matching of the input-output sets is an intuitive way. However, this is not sufficient for the following reason. In practice, it has been seen that different services are using different keywords to name a similar functionality what they provide. For instance, $Hotel\_Book$, H_Bk, H_Book, and $Book\_the\_Hotel$ are the keywords that can be used by different services for a hotel booking request. In most of the cases, these keywords are synonyms of each other. While calculating the similarity index between two services (computing similarity), synonyms can be identified using classical techniques such as text mining algorithms [189]. However, identifying the similarity between non-synonym keywords is not possible with classical search and match algorithms. In order to identify similarity among these words, one need to investigate the functionalities they provide.

Goal set based *behavioral equivalence* between two services provides the conceptual basis to verify whether the behavior of two services could be considered similar or not in terms of the functionalities they provide. In other words, we consider two services behaviorally equivalent if they provide similar functionalities.

Theorem 3. *Given two Web services $w_i \neq \epsilon$ and $w_j \neq \epsilon$ such that $\mathcal{G}_i$ and $\mathcal{G}_j$ are their goal sets respectively. Further, let us consider that $\mathcal{G}_j \subseteq \mathcal{G}_i$. Then, the traces generated by w_i ($\mathcal{T}_{w_i}$) can mimic all the behaviors represented by the traces generated by w_j ($\mathcal{T}_{w_j}$).*

Proof. It is given that $\mathcal{G}_j \subseteq \mathcal{G}_i$. Further, without loss of generality we can assume that $\mathcal{G}_j = \{w_n\}$ and w_n is a basic Web service. Since $\mathcal{G}_j \subseteq \mathcal{G}_i$, $w_n \in \mathcal{G}_i$. Let $Inv(w_n)$ be a set such that it consists of all the services that can directly invoke the service w_n. Since $w_n \in \mathcal{G}_i$, there must be a trace form w_i to w_n. Let $w_i \rightarrow\rightarrow w_n$ be a representation for trace from w_i to w_n. Similarly, we consider that there exists a trace

$w_j \rightarrow\!\!\rightarrow w_n$. For the sake of convenience, we assume that both traces are maximal. Suppose there exists Web services w_k and w_l such that w_k lies on the trace $w_i \rightarrow\!\!\rightarrow w_n$ and invokes w_n (i.e. $w_i \rightarrow\!\!\rightarrow w_k \rightarrow w_n$) and w_l lies on the trace $w_j \rightarrow\!\!\rightarrow w_n$ and invokes w_n (i.e. $w_j \rightarrow\!\!\rightarrow w_l \rightarrow w_n$). This implies $\{w_k, w_l\} \in Inv(w_n)$. Further, it infers that w_l lies somewhere on $w_i \rightarrow\!\!\rightarrow w_k \rightarrow w_n$ and w_k lies somewhere on $w_j \rightarrow\!\!\rightarrow w_l \rightarrow w_n$ as if two composite services invoke a basic service for similar functionality, then they can invoke each other. By repeating this process for all the services in $w_j \rightarrow\!\!\rightarrow w_n$, one can find that all the participating services of $w_j \rightarrow\!\!\rightarrow w_n$ appear in $w_i \rightarrow\!\!\rightarrow w_n$ only with a different permutation. Hence, $w_i \rightarrow\!\!\rightarrow w_n$ mimics the behavior represented by the trace $w_j \rightarrow\!\!\rightarrow w_n$. □

Corollary 4. *Expression 4.4 assures that the traces generated by w_i ($\mathcal{T}_{w_i}$) are partially equivalent to the traces generated by w_j ($\mathcal{T}_{w_j}$) and vice versa*

$$\exists w_i, w_j \in \mathcal{W} : (w_i \rightsquigarrow \mathcal{G}_i) \wedge (w_j \rightsquigarrow \mathcal{G}_j) \wedge (\mathcal{G}_i \cap \mathcal{G}_j \neq \emptyset) \tag{4.4}$$

Proof. By using Theorem 3, the proof of this corollary is direct and hence omitted. □

Corollary 5. *Expression 4.5 infers that the goal set based complete behavioral equivalence exists between w_i and w_j.*

$$\exists w_i, w_j \in \mathcal{W} : ((w_i \rightsquigarrow \mathcal{G}_i) \wedge (w_j \in \mathcal{G}_i)) \wedge \\ ((w_j \rightsquigarrow \mathcal{G}_j) \wedge (w_i \in \mathcal{G}_j)) \wedge (\mathcal{G}_i \backslash w_j = \mathcal{G}_j \backslash w_i)) \tag{4.5}$$

Proof. By using Theorem 3, the proof of this corollary is direct and hence omitted. □

The condition stated in Corollary 5 assures that for a trace $T_p \in \mathcal{T}_{w_i}$, there exists a trace $T_q \in \mathcal{T}_{w_j}$ such that T_p can replace T_q, and conversely, T_q can also replace T_p.

4.2.3 Deadlock Verification

In this subsection, we study the detection of possible communication deadlocks in a set of available services.

Theorem 6. *Given a set of services $\mathcal{W}$ and two Web services $w_i, w_j \in \mathcal{W}$ such that $P(\mathcal{G}_i)$ and $P(\mathcal{G}_j)$ are the parent sets of the goal sets of w_i and w_j. Then, there exists a communication deadlock between w_i and w_j if and only if $w_i \in P(\mathcal{G}_j)$ and $w_j \in P(\mathcal{G}_i)$.*

Proof. In a Web services composition scenario only communication deadlock is possible, where messages (request or reply) are considered as the resources. In order to arise a deadlock situation, there are four necessary and sufficient conditions: mutual exclusion, hold and wait, no preemption, and circular wait. Following is the analysis of fulfillment of all the four conditions in the context of Web services with an assumption that $w_i \in P(\mathcal{G}_j)$ and $w_j \in P(\mathcal{G}_i)$:

- **Mutual exclusion:** The concept of mutual exclusion is always being fulfilled in the context of Web services as sharing is not possible for messages and a message gets generated uniquely based on a particular request and intended for that request only.

- **Hold and wait:** This condition occurs if a service has received a request from another service and not replying back because the service itself is waiting to hear a reply from another service. Assuming that this condition is true in a RCG, it does not affect the properties and generality of the RCG.

- **No-preemption:** Preemption is not possible in a Web service composition scenario as a communicated message cannot be taken back.

- **Circular wait:** Since $w_i \in P(\mathcal{G}_j)$, w_i exists somewhere in the RCG ignited by w_j. As w_i is a composite service, it has to invoke other services to accomplish the task. The RCG generated by considering w_i as an igniter service provides all the possible composition scenarios. Since it has been provided that $w_j \in P(\mathcal{G}_i)$ (that is, w_j exists somewhere in the RCG ignited by w_i), the condition of circular wait occurs in the RCGs if w_i in $P(\mathcal{G}_j)$ is waiting for a reply message from the next service in its chain and w_j in $P(\mathcal{G}_i)$ is waiting for a reply message from the next service in its chain.

Based on the said analysis, we find that mutual exclusion, hold and wait, no preemption, and circular wait are being fulfilled if $w_i \in P(\mathcal{G}_j)$ and $w_j \in P(\mathcal{G}_i)$. Therefore, it is proved that the communication deadlock exists between w_i and w_j. $\square$

The described Web service composition verification process is presented in Algorithm 4.3. This algorithm takes inputs as a set of Web services $\mathcal{W}$, and it returns the possible deadlock conditions and the behavioral equivalences between services.

4.3 Illustrative Scenario

This section describes the process for verification of Web service composition by means of a classical travel agency scenario. Participant services and their respective input output sets of messages are described in Chapter 3.

Algorithm 4.3 verifies all the possible composition scenarios that could be formed among the given Web services. The algorithm first computes the set of goal sets $\mathbb{G}$ for each service in $\mathcal{W}$ as follows.

$$\begin{aligned}
TA_1 &\rightsquigarrow \mathcal{G}_{TA_1} = \{TA_2, FB, HB_1, HB_2, EQ\} \\
TA_2 &\rightsquigarrow \mathcal{G}_{TA_2} = \{TA_1, FB, HB_1, HB_2, CB\} \\
HB_1 &\rightsquigarrow \mathcal{G}_{HB_1} = \{TA_1, TA_2, HB_2\} \\
HB_2 &\rightsquigarrow \mathcal{G}_{HB_2} = \{TA_1, TA_2, HB_1\} \\
FB &\rightsquigarrow \mathcal{G}_{FB} = \{TA_1, TA_2, \} \\
CB &\rightsquigarrow \mathcal{G}_{CB} = \{CB\} \\
CT &\rightsquigarrow \mathcal{G}_{CT} = \{CT\} \\
EQ &\rightsquigarrow \mathcal{G}_{EQ} = \{EQ\} \\
CS &\rightsquigarrow \mathcal{G}_{CS} = \{EQ\}
\end{aligned}$$

The set of goal sets is:

$$\mathbb{G} = \{\mathcal{G}_{TA1},\ \mathcal{G}_{TA2},\ \mathcal{G}_{HB1},\ \mathcal{G}_{HB2},\ \mathcal{G}_{FB},\ \mathcal{G}_{CB},\ \mathcal{G}_{CT},\ \mathcal{G}_{EQ},\ \mathcal{G}_{CS}\}$$

Further, it computes the set of non-reachable services $S_{nr} = \{CT\}$, set of strict igniter services $S_{ig} = \{CS\}$, and set of strict terminator services $S_{tm} = \{CB,\ EQ\}$.

Algorithm 4.3 WS-Composition-Verification($\mathcal{W}$)

```
    Input: $\mathcal{W} = \{w_1, \cdots, w_m, \epsilon\}, \succ$
    Output: Behavioral equivalences, deadlock conditions
 1: $\mathbb{G} \leftarrow \emptyset$                                   ▷ let $\mathbb{G}$ be an empty set
 2: for all $w_i \in \mathcal{W}$ do
 3:     $w_i \leadsto \mathcal{G}_i$                                    ▷ $w_i$ leads to the goal set $\mathcal{G}_i$
 4:     $\mathbb{G} \leftarrow \{\mathbb{G} \cup \mathcal{G}_i\}$       ▷ $\mathbb{G}$ becomes set of goal sets of all the services in $\mathcal{W}$
 5: end for
 6: $S_{tm} = \emptyset$, $S_{nr} = \emptyset$, $S_{ig} = \emptyset$   ▷ $S_{tm}$, $S_{nr}$, and $S_{ig}$ are the empty sets
 7: for all $\mathcal{G}_i \in \mathbb{G}$ do
 8:     if $\mathcal{G}_i = \emptyset$ then
 9:         for all $\mathcal{G}_j \in \mathbb{G}$ do
10:             if $w_i \in P(\mathcal{G}_j)$ then          ▷ $P(\mathcal{G}_j)$ is the set of parent nodes of the services in $\mathcal{G}_j$
11:                 $S_{tm} \leftarrow \{S_{tm} \cup w_i\}$     ▷ $S_{tm}$ is the set of terminator services
12:             else
13:                 $S_{nr} \leftarrow \{S_{nr} \cup w_i\}$     ▷ $S_{nr}$ is the set of non-reachable services
14:             end if
15:         end for
16:     else if $\nexists w_j \in \mathcal{W}$ $such\ that$ $w_i \in P(\mathcal{G}_j)$ then
17:         $S_{ig} \leftarrow \{S_{ig} \cup w_i\}$             ▷ $S_{ig}$ is the set of igniter services
18:     end if
19: end for
20: $\mathcal{W} \leftarrow \{\mathcal{W} - S_{nr}\}$
21: for all $w_i \in S_{nr}$ do
22:     $w_i \leadsto \mathcal{G}_i$
23:     $\mathbb{G} \leftarrow \{\mathbb{G} - \mathcal{G}_i\}$         ▷ excluding unnecssary sets from $\mathbb{G}$
24: end for
25: for all $\mathcal{G}_i \in \mathbb{G}$ do
26:     for all $\mathcal{G}_i \in \mathbb{G}$ do
27:         if $\mathcal{G}_i = \mathcal{G}_j$ then
28:             $\mathcal{T}_{w_i}$ and $\mathcal{T}_{w_j}$ behave alike          ▷ complete behavioral equivalence
29:         else if $\mathcal{G}_i \subset \mathcal{G}_j$ then
30:             $\mathcal{T}_{w_j}$ consists of all the behavior of $\mathcal{T}_{w_i}$   ▷ behavioral subsumption
31:         else if $\mathcal{G}_i \cap \mathcal{G}_j \neq \emptyset$ then
32:             behavior of $\mathcal{T}_{w_i}$ is partially equivalent to $\mathcal{T}_{w_j}$   ▷ partial behavioral equivalence
33:         else if $(w_j \in \mathcal{G}_i) \wedge (w_i \in \mathcal{G}_j) = True$ then
34:             deadlock is possible in $\mathcal{T}_{w_i}$ and $\mathcal{T}_{w_j}$   ▷ deadlock condition
35:         end if
36:     end for
37: end for
```

Since CT is found as a non-reachable service, the proposed algorithm excludes it from the set $\mathcal{W}$. By excluding the non-reachable service and its respective goal set, the algorithm reforms the set $\mathbb{G}$ to a minimal set. Since $\mathcal{G}_{CS} \subset \mathcal{G}_{TA_1}$, the obligations of CS can be achieved with the help of TA_1. Since $\mathcal{G}_{TA_1} \cap \mathcal{G}_{TA_2} = \{FB, HB_1, HB_2\}$, TA_1 and TA_2 are partially equivalent (they show common behavior for some inputs). FB and TA_1 OR TA_2 are also partially equivalent (as $FB \cap TA_1 = TA_2$ and $FB \cap TA_2 = TA_1$). As per the results of the algorithm, HB_1 and HB_2 are completely equivalent (they fulfill the same accomplishments). HB_1 can be completely substituted with HB_2 and vice versa. Furthermore, the algorithm infers that deadlock conditions are possible in the traces generated by TA_1 and TA_2 and in the traces generated by HB_1, HB_2, and FB. One possible pattern of deadlock is $TA_1 \rightarrow HB_1 \rightarrow TA_2 \rightarrow TA_1$. Thus, all behavioral equivalences and deadlock conditions possible in compositions among services in $\mathcal{W}$ are recognized by the using the WSSP graph and deduced logical inferences.

We developed an integrated tool for the verification of Web services composition, interaction, and compositional equivalence. The description of this integrated tool is presented in Chapter 7.

4.4 Related Work

The model checking based Web services composition verification techniques [11, 12, 13, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 67, 68, 69, 70, 71, 76, 77] are described in detail in Chapter 2. Therefore, in this section, these papers are not described again.

A Web service is a reactive system; it consumes messages from the environment and produces answers depending on its internal state. The question of equivalence investigates whether a Web service can be replaced by another while the remaining components stay untouched. Martens [114] advocated that an adequate notion of simulation or equivalence, first of all, should be derived semantically from the field of application and he defined equivalence between two services as follows:

Definition 4.4.1. (Simulation/Equivalence) A workflow module A simulates a workflow module B if each utilizing environment of module B is an utilizing environment of module A, too. Two workflow modules A and B are called equivalent, if the module A simulates the module B and the module B simulates the module A.

In this approach, the verification of equivalence is based on a formal simulation relation between the communication graphs of both workflow modules – the explicit representation of the module's externally visible behavior.

Researchers used process algebras [35, 190] as a family of languages for formally describing the behavior of Web services [37, 184, 191]. In order to compare the behavior of two Web services, classical phenomenon such as trace equivalence and bisimulation [35] can be used. In trace equivalence, two communicating systems could be compared by looking from the outside only at the sequences of messages (called traces) being exchanged. However, trace equivalence is too weak to identify equivalence between Web services [192]. In bisimulation equivalence, one compares the internal states and enabled the transitions of both given modules: A state z_s simulates a state z if each transition (identified by its label), which is enabled in z, is also enabled in z_s, and the reached state z'_s simulates the reached state z'. Two modules are equivalent if their initial states simulate each other. However, the requirement of bisimulation equivalence is too restrictive in the context of Web services [114]. Therefore, based on this discussion, we can deduce that the classical trace equivalence and bisimulation theory cannot be applied in the context of Web service composition.

4.5 Summary

In this chapter, we described a RCA-based formal verification methodology for Web services composition. It partitioned the candidate Web services being considered for composition into several subsets on the basis of service invocation order. Arranging these subsets in a specific fashion results in a WSSP graph. Using a WSSP graph satisfying disjointness and orthogonality, the possible deadlock scenarios can be detected and resolved in Web services composition. Misbehaving Web services workflow can also be investigated by using the concept of WSSP graph and goal set.

Chapter 5

Web Service Interaction Verification Using RCA

In this chapter, we describe a RCA based modeling and verification technique for Web service interaction. Given a set of Web services, by using RCA (described in Chapter 3), we generate a *recursive composition interaction graph* (RCIG). In order to capture the requirement specifications of a Web service interaction scenario, we use *recursive composition specification language* (RCSL) as a requirement specification language. Further, we employ the RCIG as an interpretation model to interpret the semantics of a RCSL formula.

5.1 Recursive Composition Interaction Graph (RCIG)

5.1.1 RCIG Formation

Given a set of Web services $\mathcal{W}$ and an input argument such as a message $\langle I_p \rangle$ or a service $\langle w_i \rangle$ or a service-message tuple $\langle w_i, I_p \rangle$, the application of recursive composition forms a graph. We call it as a *recursive composition interaction graph* (RCIG) (see definition 5.1.1). Algorithm 5.1 describes a mechanism of forming a RCIG using recursive composition. Algorithm 5.1 accepts $\mathcal{W}$ and I_p as inputs and produces a RCIG rooted at $\langle I_p \rangle$ as the output. A RCIG is a directed acyclic graph (DAG) constructed in depth first order, in which, several nodes may have similar values. In the graph, every node is assigned with a unique identification number ($myNodeNo$) so that a node can be distinguished from other nodes irrespective of their content. Further, Algorithm 5.1 ensures that a node does not appear more than once in a path from the root node to a terminal node. Distinguishing a node from other nodes is important as whenever a new node is created, it has to be associated with its parent node using a directed edge.

Algorithm 5.1 employs two functions: (i) RootNode($\mathcal{W}, I_p$) constructs the rootnode of a RCIG and (ii) SubGraph($\mathcal{W}, I_p, parentNodeNo, seqMsgStack$) (written as algorithm 5.2) constructs the rest of the graph. SubGraph() is a recursive function that implements the recursive composition operation. Whenever a node has to be created, a function MakeNode() is called. MakeNode() is provided with the following parameters: *parentNode*, *serviceName*, *myNodeNo*, *parentNodeNo*, and *messageName*. MakeNode() fills every node with the text content: either a service-message tuple or a service-response tuple and a node number: *myNodeNo*. Further, MakeNode() connects the node with the *parentNode* by a directed edge. Algorithm 5.1 does not detail MakeNode() as it is concerned with graph drawing processes.

Definition 5.1.1 (Recursive composition interaction graph (RCIG))**.** A RCIG is a tuple $\langle V, E\rangle$, where V is a set of nodes (either in service-input format or in service-response format) and E is a set of directed edges. An edge connects a node with a set of nodes ($E : V \to 2^V$) such that the following condition holds

$$E(v_i) = U, \text{where } v_i \in V \text{ and } U \subseteq V \text{ iff } \forall v_j \in U : v_j \in \succ_R (v_i) \qquad (5.1)$$

In the literature, interactions among services are defined and handled in many ways [15, 91, 92] based on their modeling approaches. In our context, we use the term *Trace* to name an interaction pattern from the RCIG, and we represent it using the letter "T". A trace is formally defined as follows:

Definition 5.1.2 (Trace)**.** A trace T is a RCIG such that a node in the graph can have only one child utmost.

Algorithm 5.1 RCIGFormation($\mathcal{W}, I_p$)

```
    Input: $\mathcal{W}$ (set of Web services), $I_p$ (input message)
    Output: RCIG rooted at $I_p$

 1: $\mathcal{W}_1 \leftarrow \emptyset$                                  ▷ $\mathcal{W}$ is a set of Web services
 2: int $myNodeNo \leftarrow 0$
 3: function RootNode($\mathcal{W}$,$I_p$)
 4:     String $parent \leftarrow Null$
 5:     String $service \leftarrow Null$
 6:     int $parentNodeNo \leftarrow -1$
 7:     $\mathcal{W}_1 \leftarrow \mathcal{W}$
          //root creation
 8:     MakeNode(parent, service, myNodeNo, parentNodeNo, $I_p$)
          //rest of the graph
 9:     $parentNodeNo \leftarrow 0$
10:     parent $\leftarrow I_p$
11:     Stack $\langle Node \rangle$ seqMsgStack $\leftarrow \emptyset$          ▷ crating a stack that consists nodes
12:     int $seqNo \leftarrow 0$
13:     SubGraph($\mathcal{W}$, $I_p$, parentNodeNo, parent, seqMsgStack)
14: end function
```

Let $\mathcal{W}$ be a set of Web services, $w_i \in \mathcal{W}$, and $\mathcal{T}_{w_i} = \{T_0, T_1, \cdots, T_n\}$ represents a set which contains all the traces generated by applying the recursive composition on w_i. Similarly, $\mathcal{T}_{I_p}$ represents a set that contains all the traces generated by applying the recursive composition on I_p. For the sake of convenience, we always extract the traces from left to right in a RCIG. We follow the concept of traces, while studying the behavioral equivalence of services.

Subtrace. Let T_i and T_j be two traces. Let N_i and N_j be the set of nodes in T_i and T_j respectively. Let R_i and R_j be the relations that map a node to another node in T_i and T_j. Then, T_j is a subtrace of T_i (represented as $T_j \subset T_i$) if and only if $N_j \subset N_i$ and $R_j \subset R_i$.

There are two types of traces based on the termination condition:

Definition 5.1.3 (Open Trace). An open trace is a trace that ends with a service-

Algorithm 5.2 SubGraph($\mathcal{W}$, I_p, parentNodeNo, parent, seqMsgStack)

```
function subGraph($\mathcal{W}$, $I_p$, parentNodeNo, parent, seqMsgStack)
    int $i \leftarrow 0$
    for all $w_i \in \mathcal{W}$ do
        if $I_p \in w_i.I$ then
            myNodeNo = myNodeNo + 1
            MakeNode(parent, $w_i$, myNodeNo, parentNodeNo, $I_p$)
            $\mathcal{W}_{temp} \leftarrow \mathcal{W}/w_i$
            $\mathcal{F} \leftarrow (w_i.I_p).FwdList$    ▷ $(w_i.I_p)$.FwdList - all forward elements within $(w_i.I_p)$
            if $\mathcal{F} \neq \emptyset$ then
                int fnn $\leftarrow$ myNodeNo
                int $j \leftarrow 0$
                for all $F_j \in \mathcal{F}$ do
                    $\mathcal{S} \leftarrow F_j.SeqList$    ▷ $F_j$.SeqList - all sequential messages within $\langle F_j \rangle$
                    int $l \leftarrow 0$
                    $I_p \leftarrow S_l$    ▷ $S_l \in \mathcal{S}$
                    while $S_{l+1} \neq Null$ do
                        $l \leftarrow l + 1$
                        seqMsgStack.push($S_{l+1}$)    ▷ pushing $S_{l+1}$ is in seqMsgStack
                    end while
                    subGraph($\mathcal{W}_{temp}$, $I_p$, fnn, $w_i$, seqMsgStack)
                end for
            else
                $R \leftarrow (w_i.I_p).ResList$    ▷ $(w_i.I_p)$.ResList - all response messages for $(w_i.I_p)$
                int resParentNodeNo $\leftarrow$ myNodeNo
                parent $\leftarrow w_i$
                int $k \leftarrow 0$
```

input tuple.

Definition 5.1.4 (Closed Trace)**.** A closed trace is a trace that ends with a service-response tuple.

For a given set of Web services $\mathcal{W}$ and an input message I_p, if T_{I_p} consists an open

```
27:             for all $R_k \in R$ do
28:                 Stack ⟨Node⟩ $st$ ← seqMsgStack ▷ copying seqMsgStack in st
29:                 $I_p \leftarrow R_k$
30:                 myNodeNo ← myNodeNo + 1
31:                 MakeNode(parent, $w_i$, myNodeNo, resParentNodeNo, $I_p$)
32:                 String keyword                ▷ keyword is undesirable string
33:                 if $I_p \neq$ keyword then
34:                     if $st \neq \emptyset$ then
35:                         Node nd ← st.pop()
36:                         String msg1 ← nd
37:                         String seqParent = nd.getOwnerDocument()
38:                         int fn ← myNodeNo
39:                         myNodeNo ← myNodeNo + 1
40:                         MakeNode(parent, seqParent, myNodeNo, fn, msg1)
41:                         $\mathcal{W}_{tsvdb} \leftarrow \mathcal{W}_1$/seqParent
42:                         int fnn ← myNodeNo
43:                         SubGraph($\mathcal{W}_{tsvdb}$, msg1, fnn, $w_i$, st)
44:                     end if
45:                 end if
46:             end for
47:         end if
48:     end if
49:   end for
50: end function
```

trace, it implies that adequate candidate services are not available in $\mathcal{W}$ to compute all the possibilities. Since an open trace is a faulty trace, it is not desirable in service composition scenarios.

There are three types of RCIG based on its formation style: service-driven, message-driven, and service-message driven. In a service-driven RCIG, a service name is the generator of the graph. The root node consists of the service name and is preceded by service-message tuples. For instance, let w_i be a service name that forms the root node. Then, all immediate nodes are of the form $\langle w_i, I_p \rangle$, where $I_p \in w_i.I$.

In a message-driven RCIG, a message name (say, I_p) is the generator of the graph. The root node consists of the message name and is preceded by service-message tuples such that all immediate nodes (after root node) are restrictive successor of the I_p. In a service-message driven RCIG, a service-message tuple (say, $\langle w_i, I_p \rangle$) is the generator of the graph. The root node consists of the service-message tuple and is preceded by service-message tuples such that all immediate nodes (after root node) are restrictive successor of the previous node.

5.1.2 Implementation of the RCIG

A WSDL document is the description of a Web service written in XML format. A WSDL document consists of the following elements: ⟨definition⟩, ⟨types⟩, ⟨message⟩, ⟨operation⟩, ⟨portType⟩, ⟨binding⟩, ⟨port⟩, and ⟨service⟩. Listing 5.1 depicts an abstract structural view of a WSDL document. The ⟨portType⟩ element combines multiple message elements to form a complete one-way or round-trip operation. WSDL supports four basic patterns of operations as: one-way, request-response, solicit-response, and notification. In order to support verification of completely automated and dynamic Web service composition, we use an intermediate representation (see Listing 5.2) that is derived from an existing WSDL structure with the following minor modifications in the ⟨operation⟩ element of the WSDL document. Except the ⟨operation⟩ element, the remaining structure of WSDL is not altered.

For a basic service, we adopt the similar structure of a classical WSDL document. However, instead of input-output set of messages, we propose the input-response set of messages in the ⟨operation⟩ element. For a composite service, within the ⟨operation⟩ element, an ⟨input⟩ element is preceded by a number of ⟨forward⟩ elements and each ⟨forward⟩ element consists of ⟨sequential⟩ elements. A ⟨sequential⟩ element is a text element that consists a message that has to be forwarded to other services. All ⟨forward⟩ elements corresponding to an input message get triggered in parallel and all sequential messages within a ⟨forward⟩ element get triggered successively in the order in which they appear. A ⟨forward⟩ element is not a text element, whereas ⟨input⟩ and ⟨sequential⟩ elements are text elements. The purpose of a ⟨forward⟩ element is to discriminate streams of parallel flows from each other.

Throughout the chapter, in our examples, wherever it is required to provide a WSDL document for a service, we provide a fraction (only ⟨portType⟩ element) of that WSDL document to avoid unnecessary complex details and to support better understanding.

Listing 5.1: WSDL Document Structure

```
<definitions>
    <types>
        definition of types
            ........
    </types>

    <message>
        definition of a message
            ....
    </message>

    <portType name...>*
        <operation name...>
            <input message.../>
            <output message.../>
        </operation>
    </portType>

    <binding>
        definition of a binding
            ....
    </binding>

    <service>
        definition of a service
            ....
    </service>
</definitions>
```

Listing 5.2: Intermediate Representation

```
<definitions>
    <types>
        definition of types
            ........
    </types>

    <message>
        definition of a message
            ....
    </message>

    <portType name...>*
        <operation name...>
            <input message.../>
                <forward...>*
                    <sequential.../>*
                </forward>
            <output message.../>
        </operation>
    </portType>

    <binding>
        definition of a binding
            ....
    </binding>

    <service>
        definition of a service
            ....
    </service>
</definitions>
```

The rest of this subsection presents an implementation (using Java) of recursive composition interaction graph (RCIG) using our classical travel agency scenario (described in Chapter 3). Let $\mathcal{W} = TA,\ HB,\ FB,\ CB,\ EQ$, and $Null$ be a finite set of Web services. $TA,\ HB,\ FB,\ CB,\ EQ,$ and $Null$ are the abbreviations for services: *travel agency, hotel booking, flight booking, car booking, Enquiry* and *Null*, respectively. TA is a composite service, FB, HB, CB, and EQ are basic services, and $Null$ is an empty service. Listings 5.3-5.8 are the simplified and fractional WSDL documents in the proposed format for the candidate Web services.

Let $\mathcal{W}$ and $\langle Travel\_Booking\rangle$ be the input arguments to construct a RCIG. Fig. 5.1, a RCIG generated by using Algorithm 5.1, shows all the possible interaction patterns in $\mathcal{W}$ triggered by the input message $Travel\_Booking$. Algorithm 5.1 creates a root node (s_0) labeled with the input argument $Travel\_Booking$. Further, it searches for the existence of the input argument $Travel\_Booking$ in the input set of available services and $Travel\_Booking$ is found only in TA (Listing 5.3 is the respective WSDL). On receiving of this input, TA initiates three parallel traces by using the recursive composition operator. These traces proceed further by recursively composing the restrictive successor services and stop when no successor is remaining to compose with.

Listing 5.3: Web Service TA

```
...
<portType...>
<input>Travel_Booking
</input>
<forward no="1">
    <sequential no="1">
        Flight_Avail?
    </sequential>
    <sequential no="2">
        Flight_Book
    </sequential>
    <sequential no="3">
        Hotel_Avail?
    </sequential>
    <sequential no="4">
        Hotel_Book
    </sequential>
</forward>
<forward no="2">
    <sequential no="1">
        City_Name
    </sequential>
</forward>
<forward no="3">
    <sequential no="1">
        Car_Avail?
    </sequential>
    <sequential no="2">Car_Book
    </sequential>
</forward>
<response>
    Travel_Booking_Info
</response>
</portType>
...
```

Listing 5.4: Web Service HB

```
...
<portType...>
    <input>Hotel_Avail?
    </input>
    <response no="1">
    Hotel_Yes</response>
    <response no="2">
    Hotel_No</response>
</portType>

<portType...>
    <input>Hotel_Book
    </input>
    <response no="1">
    5Star</response>
    <response no="2">
    4Star</response>
    <response no="3">
    H_Not_Booked</response>
</portType>
...
```

Listing 5.5: Web Service FB

```
...
<portType ...>
    <input>Flight_Avail?
    </input>
    <response no="1">
     Flight_Yes</response>
    <response no="2">
     Flight_No</response>
</portType>

<portType ...>
    <input>Flight_Book
    </input>
    <response no="1">
     BusinessClass</response>
    <response no="2">
     Flight_Not_Booked
    </response>
    <response no="3">
     EconomyClass</response>
</portType>
...
```

Listing 5.6: Web Service CB

```
...
<portType ...>
    <input>Car_Avail?
    </input>
    <response no="1">Car_Yes
    </response>
    <response no="2">Car_No
    </response>
</portType>

<portType>
    <input>Car_Book
    </input>
    <response no="1">Car_Booked
    </response>
    <response no="2">
         Car_Not_Booked
    </response>
</portType>
...
```

Listing 5.7: Web Service EQ

```
...
<portType ...>
    <input>City_Name
    </input>
    <response no="1">
         Whether_Condiition
    </response>
</portType>
...
```

Listing 5.8: Null Web Service

```
...
<portType>
    <input> </input>
</portType>
...
```

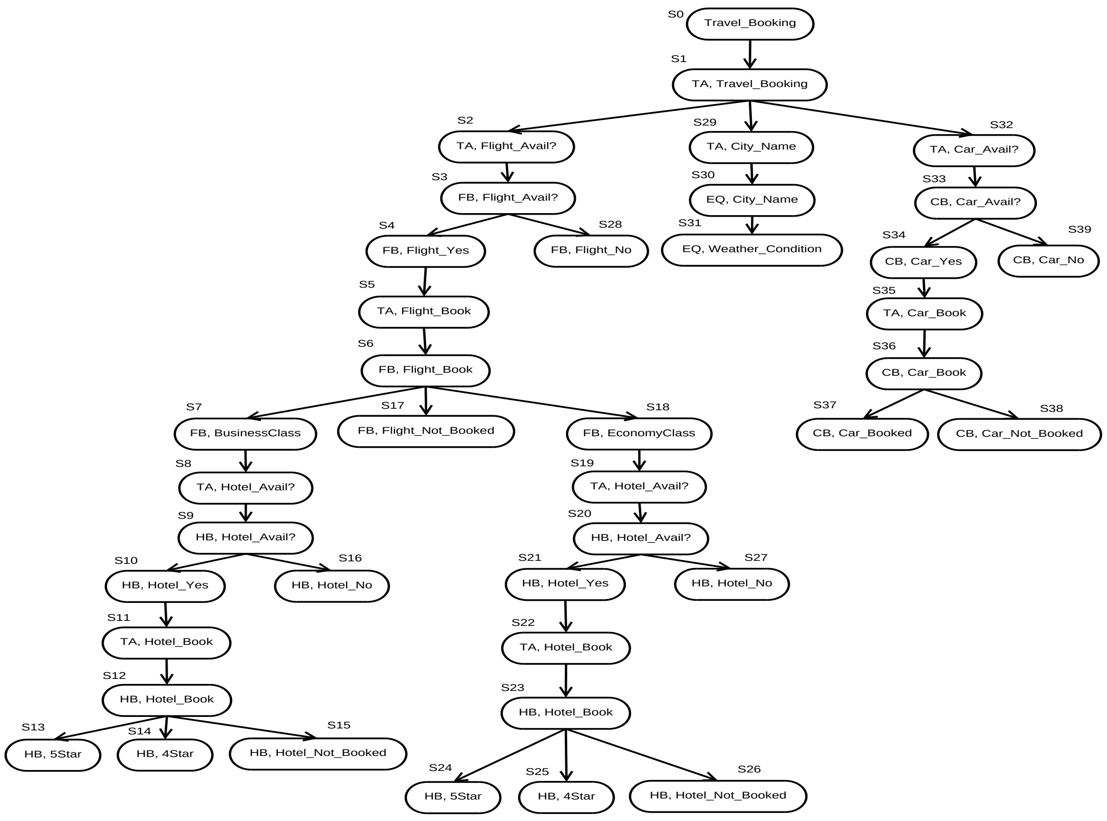


Figure 5.1: The RCIG Generated by the Input $Travel\_Booking$

5.2 Web Service Interaction Specification

The requirements of Web service interaction are classified into functional and non-functional [193]. In this chapter, our proposed verification technique exclusively focuses on functional requirements and verifies both aspects of functional requirements: *safety properties* (describe what must not happen) and *liveness properties* (describe what must happen) [194].

Throughout the chapter, M represents an interpretation model and s_i, where $i \in \mathbb{N}$, represents i^{th} node or i^{th} state (depending on the context) in M. The logical statement $M, s_0 \models \phi$ infers that a state s_0 in the model M satisfies the requirement specification ϕ. Messages with similar name can exist in several Web services. While referring a message in particular, a service name is used as prefix and to refer a message in general, a message name itself appears without any prefix. For instance, let

$\phi = w_i.m_p \rightarrow m_q$ be a specification formula. In ϕ, $w_i.m_p$ refers a message m_p in w_i and m_q is a message name in general. The specification formula ϕ infers that *if m_p is triggered from the service w_i then m_q will be triggered eventually.*

5.2.1 A Model for the Interpretation of the Semantics of a Specification Formula

RCIG is employed as a model for interpreting requirement specification formulas. The interaction between two Web services can be anticipated very easily with the help of RCIG as it explores all the possible interaction patterns. A RCIG is a graph and each branch from the root to a terminal node is considered as a trace. An interaction pattern evolves with time. However, time ordering cannot be established between two nodes that belong to two different traces in a RCIG.

In the context of Web service interaction, the communication messages are considered as propositions [15]. The truth value of a message infers whether the message has been communicated or not. For instance, if $w_i.I_p = \top$, then the Web service w_i has communicated the message I_p, otherwise not. In a trace, once truth value for a message is set true, it cannot be altered in the subsequent nodes in that trace.

In order to represent the fact that all the traces generated from a node s_i in a RCIG model M satisfies the specification formula ϕ, we write $M, s_i \models \phi$. The logical statement $M \models \phi$ infers that all the traces generated from the root node satisfies the formula ϕ and logical statement $M, s_i, T_j \models \phi$ infers that the trace T_j generated from the node s_i in the model M satisfies the formula ϕ.

5.2.2 Recursive Composition Specification Language

In order to specify the requirements regarding Web services interactions, we propose a specification language: *recursive composition specification language* (RCSL).

Definition 5.2.1 (Syntax of RCSL). RCSL has the following syntax given in Backus-Naur form:

$$\Phi ::= \top \mid \bot \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \mid (\phi \cup \phi) \mid A\phi \mid E\phi \tag{5.2}$$

where p is any propositional atom from some set of atoms and each occurrence of ϕ to the right of $::=$ stands for any already constructed formula. $\top$ and $\bot$ are well formed formulas "the tautology" and "the falsum" respectively. $\neg$, $\wedge$, $\vee$, and $\rightarrow$ are sentential connectives and are used in their usual meaning. $\cup$ is a temporal modality called *until*. A and E are path quantifiers. A stands for *all paths* and E stands for *at least one path*.

Negation symbol '$\neg$' binds the most tightly. Next in the order comes $\cup$ that binds more tightly than $\vee$ and $\wedge$, and the latter two bind more tightly than $\rightarrow$. Though RCSL consists of the constructs from both LTL and CTL, neither $RCSL \subseteq LTL$ nor $RCSL \subseteq CTL$. Let $M = (S, \rightarrow, L)$ be a RCSL model, $T = s_0, \cdots, s_n$ be a trace in M, and $n(T)$ is a collection of all nodes in a trace T. $s_i \models p$ means that a node s_i consists of the proposition p. The satisfaction relation $\models$ (explaining whether T satisfies a RCSL formula) is defined as follows:

1. $T \models \top$ ($\top$ is always true).
2. $T \not\models \bot$ ($\bot$ is always false).
3. $T \models p$ iff $\exists s_i \in n(T) : s_i \models p$
4. $T \models \neg\phi$ iff $T \not\models \phi$
5. $T \models \phi_1 \wedge \phi_2$ iff $T \models \phi_1$ and $T \models \phi_2$
6. $T \models \phi_1 \vee \phi_2$ iff $T \models \phi_1$ or $T \models \phi_2$
7. $T \models \phi_1 \rightarrow \phi_2$ iff $s_i, s_j \in n(T) : (s_i \models \phi_1 \;\wedge\; s_j \models \phi_2) \wedge (i < j)$
8. $T \models \phi_1 \cup \phi_2$ iff ϕ_1 is a negative literal of the form $\neg p$ and $\big((s_i, s_j \in n(T) : s_i \models p) \wedge (s_j \models \phi_2)\big) \implies i > j$
9. $T \models A\phi_1$ iff $T_i \models \phi_1$ for all $i \geq 1$
10. $T \models E\phi_1$ iff $T_i \models \phi_1$ there exists $i \geq 1$

Difference between temporal logic and RCSL. Temporal logic is a formal system for reasoning about time whereas RCSL reasons about possible Web service interaction patterns and verifies whether an interaction pattern is possible to be formed or not with the available services. There is a fundamental difference in motivation for utilizing anyone of them. The specification requirements are the key factors to opt a language. In linear temporal logic, there is an implicit universal quantification over the computations - the paths in state space. RCSL uses both universal and existential quantifiers explicitly, but does not use temporal operators X (next), F (finally), and G (globally). RCSL does not require X, F, and G because its interpretation model RCIG is a finite and acyclic graph where no proposition can be false at later stage once it becomes true. In branching-time temporal logic, universal and existential quantifiers are used as explicit prefixes to the temporal operators such as AF, AG, etc., whereas RCSL does not require the combination of temporal operators with quantifiers.

5.3 Web Service Interaction Verification

Algorithm 5.3 initiates the verification process. It accepts a tuple as an input argument that consists of a set of Web services ($\mathcal{W}$) and a requirement specification formula (ϕ) written in RCSL. $\mathcal{W}$ is an online Web service repository available on a specific url address and ϕ is provided by a verifier. Once ϕ is available, Algorithm 5.3 calls Algorithm 5.4 by passing ϕ as an argument. Algorithm 5.4 parses ϕ, and correspondingly it generates an abstract syntax tree (AST) (written as P_ϕ) if the given formula is free from syntax errors. The word *Token* in Algorithm 5.4 represents a sequence of characters that can be treated as a single logical entity. Typical tokens are: 1) identifiers 2) keywords 3) operators 4) special symbols, and 5) constants.

Algorithm 5.3 InteractionVerification($\mathcal{W}, \phi$)

```
    Input: $\mathcal{W}$ (a set of Web services), $\phi$ (a specification formula)
    Output: $\mathcal{W} \models \phi$ or $\mathcal{W} \not\models \phi$

 1: $P_\phi \leftarrow$ ReqSpecParsing($\phi$)                      ▷ calling Algorithm 5.4
 2: $FLAG \leftarrow TRUE$
 3: Integer $i, j, p, t$
 4: $w_j.I$ : set of all input messages in $w_j$
 5: $A_\phi$ : set of atoms in $\phi$
 6: for all $\alpha_i \in A_\phi$ do
 7:     for all $w_j \in \mathcal{W}$ do
 8:         if $\alpha_i \in w_j.I$ then
 9:             $I_p \leftarrow \alpha_i$
10:             $M \leftarrow$RCIGFormation ($\mathcal{W}, I_p$)   ▷ $M$ is a model formed by Alg. 5.1
11:             for all trace $T_t \in M$ do
12:                 $FLAG \leftarrow$ Interpretation($P_\phi, T_t$)      ▷ calling Algorithm 5.5
13:             end for
14:         end if
15:     end for
16: end for
17: if $FLAG = TRUE$ then
18:     $\mathcal{W} \models \phi$                 ▷ available services satisfy the specification formula
19: else
20:     $\mathcal{W} \not\models \phi$           ▷ available services do not satisfy the specification formula
21: end if
```

Further, Algorithm 5.3 collects all the atoms from ϕ in the set A_ϕ and observes whether an atom (α_i) belongs to an input set of a service from the set $\mathcal{W}$. If α_i is found in the input set of a service, Algorithm 5.3 invokes Algorithm 5.1 by supplying arguments $\mathcal{W}$ and I_p. After completing the processing of the input arguments, Algorithm 5.1 provides a RCIG model M rooted at I_p. Then, Algorithm 5.3 extracts the traces T_t ($t \in \mathbb{N}$) from the model M one by one and calls Algorithm 5.5 for further processing by passing the arguments P_ϕ and T_t.

Algorithm 5.4 ReqSpecParsing(ϕ)

```
    Input: $\phi$: a requirement specification formula written in RCSL
    Output: $P_\phi$: an abstract syntax tree for $\phi$

 1: int $i = 0$
 2: String $Id_i \leftarrow NULL$
 3: String $Token \leftarrow NULL$
 4: String $nextToken \leftarrow NULL$
 5: String $prevToken \leftarrow NULL$
 6: for all $Token \in \phi$ do
 7:     $TokenSet \leftarrow Token$
 8: end for
 9: $Token \leftarrow TokenSet(i)$
10: while $|TokenSet| \neq 1$ $AND$ $Token \neq Id$ do
11:     if $Token =$‘$\neg$’ then
12:         if $nextToken =$‘(’ then
13:             Parenthesis($nextToken$)
14:         else
15:             $Id_i \leftarrow nextToken$
16:             Replace $nextToken$ with $Id_i$ in $\phi$
17:         end if
18:         Replace ‘$\neg Id_i$’ with ‘$Id_{i+1}$’ in $\phi$
19:         $i \leftarrow i + 1$
20:     else if $Token =$‘(’ then
21:         while $nextToken =$‘(’ do
22:             $Token \leftarrow nextToken$
23:         end while
24:         Parenthesis($Token$)
25:     else if $Token =$‘$\cup$’ then
26:         Func($Token$)
27:     else if $Token =$‘$\vee$’ then
28:         Func($Token$)
29:     else if $Token =$‘$\wedge$’ then
30:         Func($Token$)
31:     else if $Token =$‘$\rightarrow$’ then
32:         Func($Token$)
33:     else if $Token =$‘$p$’ then    ▷ $p$ is a proposition
34:         $Id_i \leftarrow$ ‘$p$’
35:         Replace ‘$p$’ with ‘$Id_i$’ in $\phi$
36:         $i \leftarrow i + 1$
37:     else if $Token = Id$ then
38:         Skip    ▷ move to next token
39:     end if
40: end while
41: function Parenthesis($Value$)
42:     String $Token \leftarrow Value$
43:     repeat
44:         $\phi_{Temp} \leftarrow Token$
45:         $Token \leftarrow nextToken$
46:     until $Token \neq$‘)’
47:     $\phi_{Temp} \leftarrow Token$
48:     Replace $\phi_{Temp}$ with $Id_i$ in $\phi$
49:     $i \leftarrow i + 1$
50: end function
51: function Func($Token$)
52:     $Id_i \leftarrow prevToken\ Token\ nextToken$
53:     Replace $prevToken\ Token\ nextToken$ with ‘$Id_i$’ in $\phi$
54:     $i \leftarrow i + 1$
55: end function
56: $AST(Id_{i-1})$    ▷ print the abstract syntax tree for $\phi$
```

Then, Algorithm 5.5 interprets P_ϕ on the provided trace T_t and returns either TRUE or FALSE, based on its computation. In case, if the return value is TRUE, then the trace T_t is a witness example, otherwise the trace T_t is a counter example. Algorithm 5.5 recursively decomposes the AST P_ϕ in subtrees and recursively decomposes the trace T_t corresponding to the subtrees until the unit-level-subtrees (smallest non-trivial subtrees) are achieved. Now, the function PInterpretation$(subtree, trace)$ in Algorithm 5.5 interprets the unit-level-subtrees over the corresponding dividend of the trace. Once these subtrees are satisfied in the trace, satisfaction of the higher level subtrees would be investigated in bottom to top fashion.

Example 5.3.1. Let the RCIG depicted in Fig. 5.1 be an interpretation model M and ϕ_1 (see Equation 5.3) be a requirement specification formula which, formally, states that in all the traces, if flight is available and booking is requested, then either flight must be booked or hotel must not be booked until flight is booked.

$$\phi_1 = A((Flight\_Yes \wedge Flight\_Book) \rightarrow (Flight\_Booked \vee (\neg Hotel\_Booked \cup (Flight\_Booked)))) \quad (5.3)$$

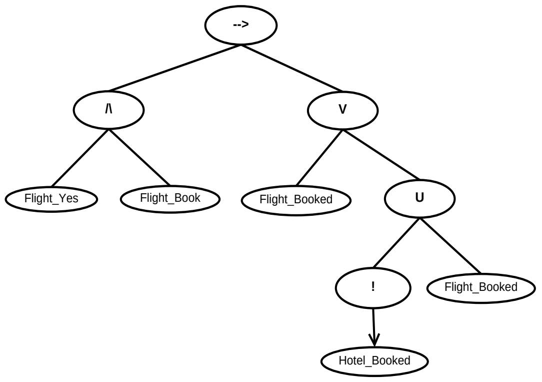


Figure 5.2: AST for the Specification Formula ϕ_1 (Eqn. 5.3)

Now, a verifier has to verify the model M against the formula ϕ_1. According to the verification technique, traces in model M are considered one by one for verification. Let us consider that a trace T_t (shown in Fig. 5.3) from the model M ($T_t = s_0, \cdots, s_{21}$) has to be verified against ϕ_1. The AST (P_{ϕ_1}) for ϕ_1 is given in Fig. 5.2.

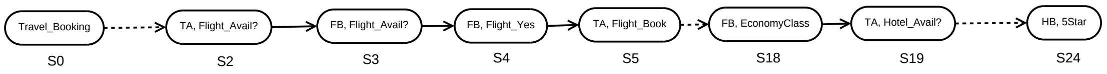


Figure 5.3: A Trace from the RCIG in Fig. 5.1

According to Algorithm 5.3, P_{ϕ_1} gets recursively decomposed in its respective subtrees until the unit-level-subtrees are achieved (see Fig. 5.4) (further no decomposition is possible). Initially, P_{ϕ_1} is decomposed into subtrees: *ST1* and *ST2*. *ST1* is decomposed into *ST11* and *ST12* that are unit-level-subtrees. Therefore, they cannot be decomposed further. *ST2* is further decomposed into: *ST21* and *ST22* and so on. Once the given formula ϕ is completely decomposed in its constituent unit-level-subtrees, the decomposition process stops.

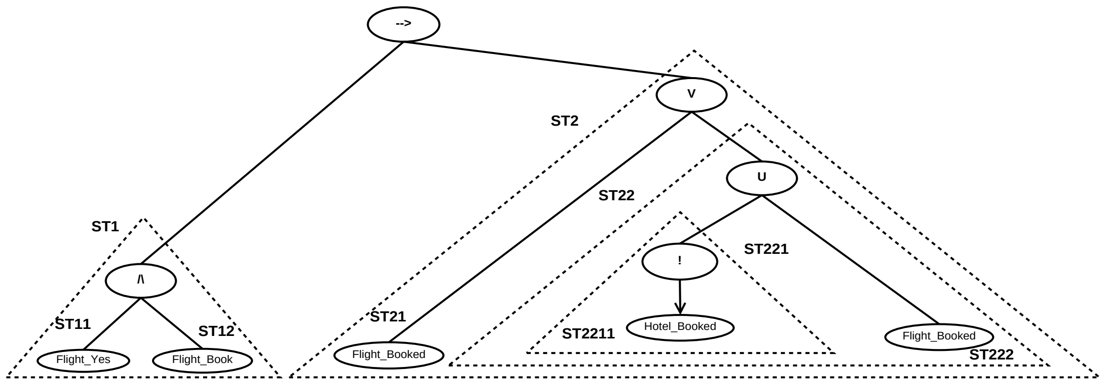


Figure 5.4: Subtree Decomposition of the AST Given in Fig. 5.2

Now, $ST1 \rightarrow ST2$ will be interpreted on the trace T_t. The subtree $ST1$ represents a subformula $Flight\_Yes \wedge Flight\_Book$ that is satisfied in the subtrace $s_0, \cdots, s_6$. Then, the trace T_t is decomposed into subtraces: $s_0, \cdots, s_6$ and $s_7, \cdots, s_{24}$. After this, the subtree $ST2 ::= ST21 \vee ST22$ $(Flight\_Booked \vee (\neg Hotel\_Booked \cup Flight\_Booked))$ gets interpreted over the subtrace $s_7, \cdots, s_{24}$. Since the subtrace satisfies $ST21$, $ST2 ::= ST21 \vee ST22$ becomes satisfied. Consequently, the requirement specification $\phi_1 = (Flight\_Yes \wedge Flight\_Book) \rightarrow (Flight\_Booked \vee (\neg Hotel\_Booked \cup Flight\_Booked))$ is satisfied in the trace T_t. Similarly, we check the satisfiability of ϕ_1 over every trace in the model M and find satisfied. Hence, $M \models \phi_1$.

Algorithm 5.5 INTERPRETATION(P_ϕ, T)

```
Input: $P_\phi$ (parse tree) and $T$ (trace)
Output: $TRUE$ or $FALSE$

1: $Root \leftarrow Root(P_\phi)$
2: $LST \leftarrow LeftSubTree(P_\phi)$
3: $RST \leftarrow RightSubTree(P_\phi)$
4: if $Root \notin \{\neg, \wedge, \vee, \rightarrow, \cup\}$ then
5:     if PINTERPRETATION$(Root, T) = TRUE$ then
6:         Return $TRUE$
7:     else
8:         Return $FALSE$
9:     end if
10: else if $Root$ = '$\neg$' then
11:     if PINTERPRETATION$(LST, T) = TRUE$ then
12:         Return $FALSE$
13:     else
14:         Return $TRUE$
15:     end if
16: else if $Root$ = '$\wedge$' then
17:     if INTERPRETATION $(LST, T) = TRUE$ AND INTERPRETATION $(RST, T) = TRUE$ then
18:         Return $TRUE$
19:     else
20:         Return $FALSE$
21:     end if
22: else if $Root$ = '$\vee$' then
23:     if INTERPRETATION $(LST, T) = TRUE$ OR INTERPRETATION $(RST, T) = TRUE$ then
24:         Return $TRUE$
25:     else
26:         Return $FALSE$
27:     end if
28: else if $Root$ = ' $\rightarrow$' then
29:     $FLAG \leftarrow FALSE$
30:     $Temp \leftarrow \emptyset$
31:     $T_{Temp} \leftarrow \emptyset$
32:     for all $node\ n \in T$ do
33:         $Temp \cdot n$   ▷ Concatenating $n$ to the existing sequence of nodes in $Temp$
34:         if INTERPRETATION $(LST, n) = TRUE$ then
35:             $FLAG \leftarrow TRUE$
36:             BREAK
37:         end if
38:     end for
39:     $T_{Temp} \leftarrow \{T - Temp\}$
40:     if INTERPRETATION $(RST, T_{Temp}) = TRUE$ then
41:         Return $TRUE$
42:     else if $FLAG = FALSE$ AND INTERPRETATION $(RST, T_{Temp}) = FALSE$ then
43:         Return $TRUE$
44:     else
45:         Return $FALSE$
46:     end if
```

```
else if $Root$ = '$\cup$' then
    $FLAG \leftarrow FALSE$
    $Temp \leftarrow \emptyset$
    for all $node\ n \in T$ do
        $Temp \cdot n$
        if INTERPRETATION ($LST'$, $Temp$) = $FALSE$ then
            $FLAG \leftarrow FALSE$
            BREAK
        end if
    end for
    if $FLAG \neq FALSE$ then
        Return $FALSE$
    else if INTERPRETATION ($RST$, $Temp$) = $TRUE$ then
        Return $FALSE$
    else
        Return $TRUE$
    end if
end if
function PINTERPRETATION(p,T)
    $FLAG \leftarrow FALSE$
    for all $node\ n \in T$ do
        if $p \in L(n)$ then        ▷ $L(n)$ means label of node $n$
            $FLAG \leftarrow TRUE$
            BREAK
        end if
    end for
    if $FLAG = TRUE$ then
        Return TRUE
    else
        Return FALSE
    end if
end function
```

5.4 Implementation and Analysis

We implemented our proposed framework of Web service interaction modeling and verification using Java programming language. Given a set of WSDL documents of candidate services, the framework accepts the following inputs:

1. An input message (I_p) or a service name (w_i) or an input-service tuple ($\langle w_i, I_p \rangle$)
2. A specification formula (ϕ) written in RCSL

Provision of an input (I_p or w_i or $\langle w_i, I_p \rangle$) generates a RCIG (say M) for interactive trace visualization and performance analysis, whereas provision of a RCSL formula (say ϕ) triggers verification process ($M \overset{?}{\models} \phi$) along with trace visualization. If a model M does not satisfy ϕ ($M \not\models \phi$), then a counter trace (T) is also generated. In the implementation, a RCIG is generated automatically using GraphViz [195] by invoking system level commands internally.

5.4.1 Performance Analysis of RCIG

The order of a graph (the total number of nodes in a graph ($|G(V)|$)) generated by a technique is an important criteria to determine the feasibility for real world implementability of a technique. The computational resources such as time and space are directly proportional to the number of nodes. In this section, we analyze how the cardinality of forward, sequential, and response messages in services affect the order of a RCIG.

5.4.1.1 Experimental Setup

Let us consider a set of three services ($\mathcal{W} := w_f, w_s, w_r$), where w_f and w_s are composite services and w_r is a basic service. All three services accept only one input message: $Hotel\_Avail$. The composite services (w_f and w_s) forward the input message to other services, whereas, the basic service (w_r), upon reception of this message, replies with the available hotel booking options. Initially, each service consists of only one output message (output represents to $\langle forward \rangle$, $\langle sequential \rangle$, and $\langle response \rangle$ elements). As per the requirement of experiment, we gradually increase the number of forward, sequential, and response messages in the services. At any stage of experiment, all output messages in w_f are parallel to each other, all output messages in w_s are sequential to each other, and all output messages in w_r are only response messages. We assume that w_f can invoke only w_s and w_s can invoke only w_r. Since w_r is a basic service, it cannot invoke any service. This assumption facilitates us with a hierarchical invocation system of services that prevents redundancy while performing recursive composition out of w_f, w_s, and w_r. All observations are taken by providing the input $\langle w_f, Hotel\_Avail \rangle$ to the RCIG construction process. The number of nodes are counted for the unfolded form of the RCIG without applying any heuristic to reduce the number of nodes.

5.4.1.2 Experimental Evaluation

There are three types of elements in the candidate services: ⟨forward⟩, ⟨sequential⟩, and ⟨response⟩. Observations are taken for the total number of nodes by increasing an element type. In order to support symmetrical growth, we assume that a

⟨forward⟩ element can be increased only in w_f, a ⟨sequential⟩ element can be increased only in w_s, and a ⟨response⟩ element can be increased only in w_r. The minimum threshold for the count of all element types is one and maximum threshold is six. The maximum threshold is set to six, that is sufficiently large to capture and observe the patterns of the growth of order of the RCIG. For taking the observations, we increase the count for an element type while keeping the count for other element types as constant at a pre-specified value.

Following are the three sets of observations corresponding to the increment of response elements, sequential elements, and forward elements respectively. Three keywords are used in the observation tables: Res, Seq, and Fwd. These keywords stand for $\langle response \rangle$, $\langle sequential \rangle$, and $\langle forward \rangle$ elements respectively. A number that precedes an element keyword is the count for that element in the respective service. For instance, '4Res 5Seq 6Fwd' indicates that there are four response elements in w_r, five sequential elements in w_s, and six forward elements in w_f.

Effect of incrementing response messages on the order of the RCIG: Table 5.1 depicts the various observations taken for total number of nodes with respect to the increment in ⟨response⟩ elements while the count of ⟨forward⟩ and ⟨sequential⟩ elements are kept constant at the values 1, 2, 3, 4, 5, and 6.

Messages	# Nodes	Messages	# Nodes	Change	# Nodes
1Res 1Seq 1Fwd	5	1Res 3Seq 3Fwd	29	1Res 5Seq 5Fwd	77
2Res 1Seq 1Fwd	6	2Res 3Seq 3Fwd	86	2Res 5Seq 5Fwd	622
3Res 1Seq 1Fwd	7	3Res 3Seq 3Fwd	197	3Res 5Seq 5Fwd	3027
4Res 1Seq 1Fwd	8	4Res 3Seq 3Fwd	380	4Res 5Seq 5Fwd	10231
5Res 1Seq 1Fwd	9	5Res 3Seq 3Fwd	653	5Res 5Seq 5Fwd	24524
6Res 1Seq 1Fwd	10	6Res 3Seq 3Fwd	1034	6Res 5Seq 5Fwd	62202
1Res 2Seq 2Fwd	14	1Res 4Seq 4Fwd	50	1Res 6Seq 6Fwd	110
2Res 2Seq 2Fwd	26	2Res 4Seq 4Fwd	242	2Res 6Seq 6Fwd	1514
3Res 2Seq 2Fwd	42	3Res 4Seq 4Fwd	802	3Res 6Seq 6Fwd	10922
4Res 2Seq 2Fwd	62	4Res 4Seq 4Fwd	2042	4Res 6Seq 6Fwd	49142
5Res 2Seq 2Fwd	86	5Res 4Seq 4Fwd	4370	5Res 6Seq 6Fwd	164054
6Res 2Seq 2Fwd	114	6Res 4Seq 4Fwd	8290	6Res 6Seq 6Fwd	447890

Table 5.1: Effect of Incrementing Response Messages on the Order of the RCIG

Based on the values provided by Table 5.1, Fig. 5.5 (split into two parts for better visibility) depicts six curves namely Seq1Fwd1, Seq2fwd2, Seq3Fwd3, Seq4Fwd4, Seq5Fwd5, and Seq6Fwd6. Nature of the curves in Fig. 5.5 are linear and polynomial. Seq1Fwd1 is a line ($y = x + 4$). Seq2Fwd2 is a polynomial of degree 2; Seq3Fwd3 and Seq4Fwd4 are polynomials of degree 3; Seq5Fwd5 and Seq6Fwd6 are polynomials of degree 4.

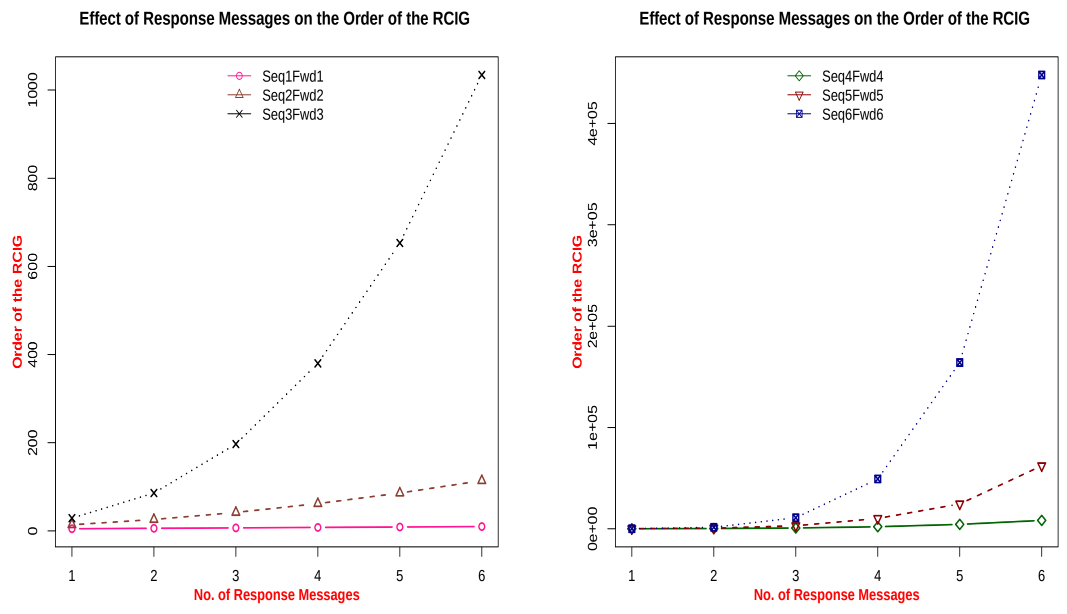


Figure 5.5: Effect of Incrementing Response Messages on the Order of the RCIG

Effect of incrementing sequential messages on the order of the RCIG: Table 5.2 depicts the various observations taken for total number of nodes with respect to the increment in ⟨sequential⟩ elements while forward and response elements are kept constant at the values 1, 2, 3, 4, 5, and 6.

Messages	# Nodes	Messages	# Nodes	Change	# Nodes
1Res 1Seq 1Fwd	5	3Res 1Seq 3Fwd	17	5Res 1Seq 5Fwd	37
1Res 2Seq 1Fwd	8	3Res 2Seq 3Fwd	62	5Res 2Seq 5Fwd	212
1Res 3Seq 1Fwd	11	3Res 3Seq 3Fwd	197	5Res 3Seq 5Fwd	1087
1Res 4Seq 1Fwd	14	3Res 4Seq 3Fwd	602	5Res 4Seq 5Fwd	5462
1Res 5Seq 1Fwd	17	3Res 5Seq 3Fwd	1817	5Res 5Seq 5Fwd	27337
1Res 6Seq 1Fwd	20	3Res 6Seq 3Fwd	5462	5Res 6Seq 5Fwd	136712
2Res 1Seq 2Fwd	10	4Res 1Seq 4Fwd	26	6Res 1Seq 6Fwd	50
2Res 2Seq 2Fwd	26	4Res 2Seq 4Fwd	122	6Res 2Seq 6Fwd	338
2Res 3Seq 2Fwd	58	4Res 3Seq 4Fwd	506	6Res 3Seq 6Fwd	2066
2Res 4Seq 2Fwd	122	4Res 4Seq 4Fwd	2042	6Res 4Seq 6Fwd	12434
2Res 5Seq 2Fwd	250	4Res 5Seq 4Fwd	8186	6Res 5Seq 6Fwd	74642
2Res 6Seq 2Fwd	506	4Res 6Seq 4Fwd	32762	6Res 6Seq 6Fwd	447890

Table 5.2: Effect of Incrementing Sequential Messages on the Order of the RCIG

Based on the values provided by Table 5.2, Fig. 5.6 depicts six curves namely Res1Fwd1, Res2Fwd2, Res3Fwd3, Res4Fwd4, Res5Fwd5, and Res6Fwd6. Nature of the curves in Fig. 5.6 are linear and polynomial. Res1Fwd1 is a line ($y = 3x + 2$). Res2Fwd2 and Res3Fwd3 are three degree polynomials. Curves Res4Fwd4 and Res5Fwd5 are four degree polynomials. Res6Fwd6 is a five degree polynomial.

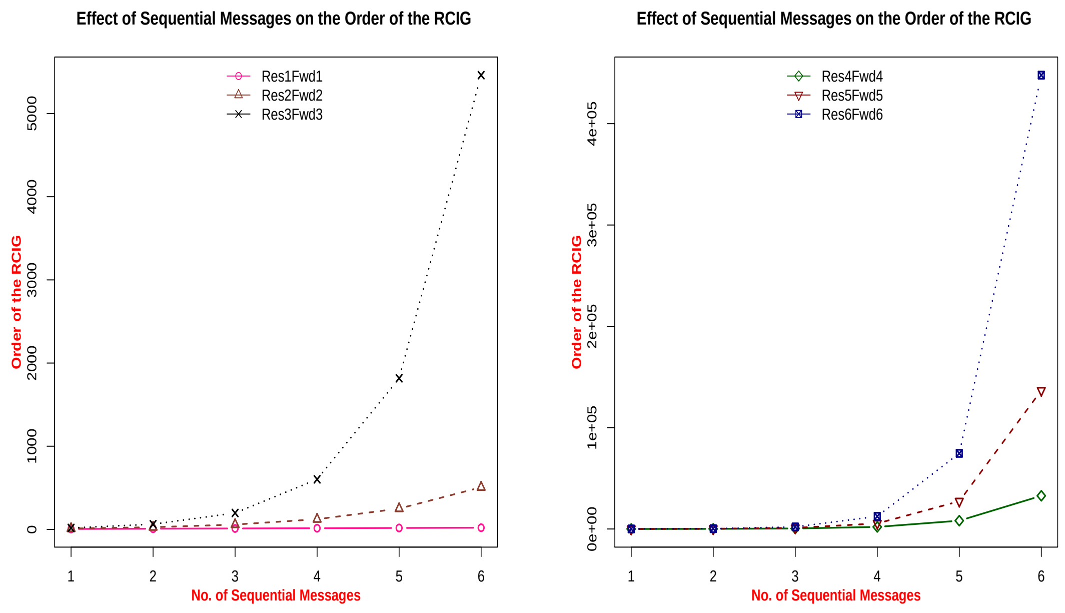


Figure 5.6: Effect of Incrementing Sequential Messages on the Order of the RCIG

Effect of incrementing forward messages on the order of the RCIG: Table 5.3 depicts the various observations taken for total number of nodes with respect to the increment in ⟨forward⟩ elements while sequential and response elements are kept constant at the values 1, 2, 3, 4, 5, and 6.

Messages	# Nodes	Messages	# Nodes	Change	# Nodes
1Res 1Seq 1Fwd	5	3Res 3Seq 1Fwd	67	5Res 5Seq 1Fwd	5467
1Res 1Seq 2Fwd	8	3Res 3Seq 2Fwd	132	5Res 5Seq 2Fwd	10935
1Res 1Seq 3Fwd	11	3Res 3Seq 3Fwd	197	5Res 5Seq 3Fwd	16403
1Res 1Seq 4Fwd	14	3Res 3Seq 4Fwd	262	5Res 5Seq 4Fwd	21870
1Res 1Seq 5Fwd	17	3Res 3Seq 5Fwd	327	5Res 5Seq 5Fwd	27337
1Res 1Seq 6Fwd	20	3Res 3Seq 6Fwd	392	5Res 5Seq 6Fwd	32804
2Res 2Seq 1Fwd	14	4Res 4Seq 1Fwd	512	6Res 6Seq 1Fwd	74650
2Res 2Seq 2Fwd	26	4Res 4Seq 2Fwd	1022	6Res 6Seq 2Fwd	149297
2Res 2Seq 3Fwd	38	4Res 4Seq 3Fwd	1531	6Res 6Seq 3Fwd	223946
2Res 2Seq 4Fwd	50	4Res 4Seq 4Fwd	2042	6Res 6Seq 4Fwd	298594
2Res 2Seq 5Fwd	62	4Res 4Seq 5Fwd	2552	6Res 6Seq 5Fwd	373242
2Res 2Seq 6Fwd	74	4Res 4Seq 6Fwd	3062	6Res 6Seq 6Fwd	447890

Table 5.3: Effect of Incrementing Forward Message on the Order of the RCIG

Based on the values provided by Table 5.3, Fig. 5.7 depicts six curves namely Res1Seq1, Res2Seq2, Res3Seq3, Res4Seq4, Res5Seq5, and Res6Seq6. Nature of the curves in Fig. 5.7 is linear.

In the best case (when count of forward elements are growing), the growth of the order of RCIG is linear. In the average case (when count of response elements are growing), the growth of the order of RCIG is lower degree polynomial, and in the worst case (when count of sequential elements are growing), the order of RCIG is a higher degree polynomial. However, in many practical cases, it is a lower degree polynomial.

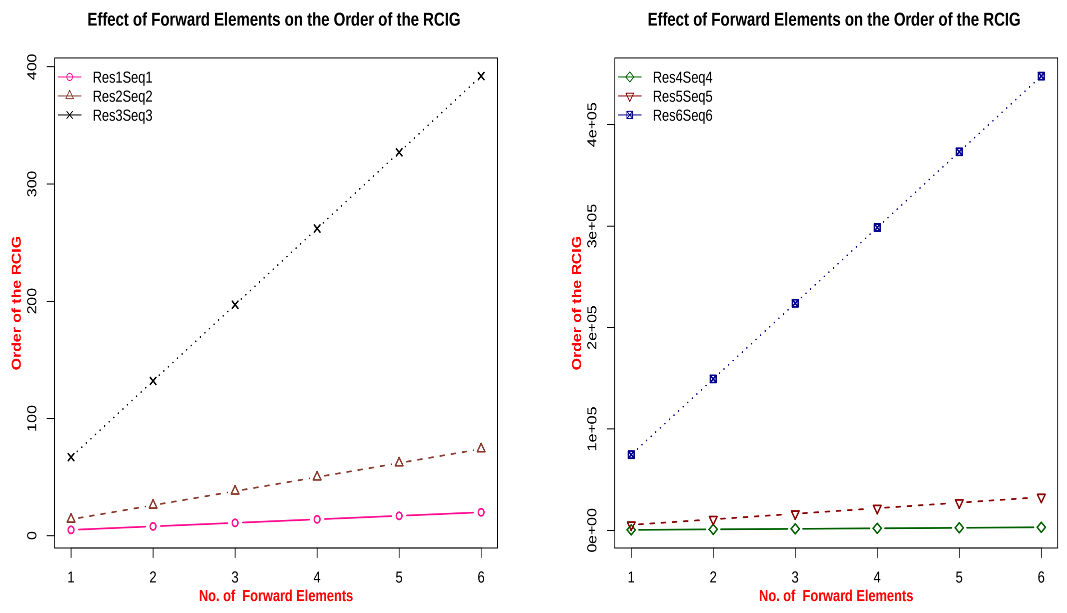


Figure 5.7: Effect of Incrementing Forward Elements on the Order of the RCIG

5.5 Related Work

This chapter is focused on service interaction modeling and verification. We compare the work presented in this chapter with the most related works from the literature on modeling and verification of service interaction.

The problem of automatic Web service composition generation is closely related to the problem of *Goal-Oriented Action Planning (GOAP)* in artificial intelligence [28, 29, 30]. In literature, several AI planning based techniques for automatic composition are available: STRIPS-based [29], PDDL-based [31], HTN-based [32], etc. Though theoretically possible, they present a number of complexities in practical implementation, such as generating and maintaining heavy amount of additional information (for instance, task library in [29]) hinders the automation process. We also use an intermediate representation in our approach, however, the conversion takes place automatically at the back-end and a verifier does not need to be concerned about it. Recent AI planning based service verification approaches, like [196] and [197], are better than previous proposals as they handle dynamic availability of services and domain-dependency of planning in more efficient way. Zou et al. [197] focused on search time reduction when finding a composite service from the Web service repository. In order to achieve their goal they converted the Web service repository into a planning domain. This transformation reduces the response time and improves the scalability of solving Web service composition problems. Kaldeli et al. [196] differed from other AI planning based techniques in that they used state variables rather than predicates as the basic elements for describing the *possible worlds* (model). From the modeling perspective, we differ from [196] as, in our approach, the possible world (model) does not need to be defined or generated by the designer explicitly. Once the specification formula is available, an interpretation model gets generated automatically. Moreover, unlike to our work, the verification aspect of Web service interaction is not discussed in [196, 197].

To form a cost-effective composite service, Jaiswal et al. [198] used a recursive composition based model. The model proposed in [198] is suitable for optimization, whereas our focus is interaction verification. Abrougui et al. [199] used recursive multi-agent systems to support dynamism in Web service composition. Like [198], this work also supports in finding a better composition solution; their motive was not to verify the interaction specifications.

The model checking based Web service interaction verification techniques [14, 15, 50, 91, 92, 94, 99, 106, 160] are described in Chapter 2. We differ from [50, 91, 92, 94, 99, 160] in that our verification technique employs possible trace-based phenomenon for verification instead of classical possible-world phenomenon and the explicit system modeling (specifications of the system provided by the designer) is not required. Further, like [14] and [106], in our proposition, the operational behavior is captured using the RCIG model and the control behavior is specified using RCSL. Moreover, in our approach, once control behavior is provided by a verifier, the related operational models are discovered automatically. El Kholy et al. [15] presented a framework to capture and verify the interactions among multi-agent based Web services. However, it can capture the conversation between two agents only if participating agents are known in advance. Moreover, it does not capture recursive composition scenarios that is done in our approach. Recently, a Petri net based formal model for verification of Web service composition [126] was proposed. However, their goal was to verify the compliance not the interaction.

5.6 Summary

In this chapter, we described modeling and verification of Web service interaction that consists of the following three steps.

1. A RCA based modeling of Web service interaction. The outcome of this modeling part is a DAG, named as RCIG, that works as an interpretation model. We found that it is completely feasible to implement the RCIG in the real-time scenarios.

2. Capturing requirements of Web service interaction scenarios using a requirement specification language RCSL that is completely interpretable on the RCIG model.

3. Verification of the given RCIG model against a user specified requirement specification (written in RCSL).

Chapter 6

Compositional Equivalence Verification between Web Service Composition Graphs Using RCA

Given a composition request, the formation of possible *Web Service Composition Graphs* (WSCGs) depends on the set of available services. Since the availability of Web services is dynamic, a new service can join or an existing service can leave the set of available services at any time instance. A change in the set may bring the structural change in a previously formed WSCG. However, this is not always the case, that a structural change in the WSCG brings the semantic change. In this chapter, our aim is to verify the compositional equivalence between two WSCGs formed before and after the structural change caused by the change in the set of available services. The solution described in this chapter is based on RCA, and by using RCA, directed acyclic WSCGs are formed for a given composition request. Then, by using WSCGs, we describe the concept of *composition expression* and *canonical composition expression*. Based on the concept of canonical composition expression, we verify compositional equivalence between two WSCGs. The advantage of this approach is that it reduces the equivalence verification to the subsumption computing between two algebraic expressions instead of directly using the WSCGs and solving a subgraph matching problem.

6.1 Composition Request and Web Service Composition Graph (WSCG)

In this section, we use RCG (described in Chapter 3) for modeling a possible composition pattern that is deadlock free and captures the recursive nature of service composition.

6.1.1 Composition Request

A composition request is a high-level composition plan [200, 201] manually created by a service designer. In this chapter, a composition request is defined by a graph (written as G_{REQ}). A G_{REQ} depicts two types of information: (i) the order of triggering of messages and (ii) the message flow specification (sequential or parallel). A composition request is classified into an *absolute composition request* (uses messages without specifying respective services) and a *relative composition request* (uses service-input tuples).

Definition 6.1.1 (Absolute Composition Request)**.** An absolute composition request is defined as an acyclic quadruple: $G_{REQ} = (N, s_0, E, \delta)$, where
(i) N is a finite nonempty set of nodes, where each node, except s_0, consists of a query or a request message.
(ii) $s_0 \in N$ is the initial node that is empty (does not consist of messages).
(iii) E is the set of directed edges connecting the nodes, each representing the computation precedence order between two nodes. For example, an edge $(s_i, s_j) \in E$ represents the precedence constraint indicating that the message in the node s_i must be processed before processing of the message in node s_j.
(iv) δ is a map which assigns a unique ordered pair of nodes to every edge.

In general, an absolute composition request is a topologically sorted DAG $G\langle V, E\rangle$ such that a vertex $v \in V$, except root node, consists of a message without an accompanying service. The advantage of an absolute composition request is that a designer can make a generic composition request, even without knowing the available services. For instance, Fig. 6.1 shows an absolute composition request which infers that flight availability must be checked followed by flight booking. Then, hotel availability must be checked followed by hotel booking. Since this is an absolute composition request, the requester services are not specified for a message.

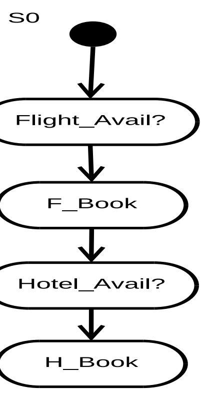


Figure 6.1: An Example of Absolute Composition Request: Sequential Message Flow

Definition 6.1.2 (Relative Composition Request)**.** A relative composition request is defined as an acyclic quadruple: $G_{REQ} = (N, s_0, E, \delta)$, where
(i) N is a finite nonempty set of nodes, where each node, except s_0, consists of a service-input tuple.
(ii) $s_0 \in N$ is the initial node that is empty (does not consist of messages).
(iii) E is the set of directed edges connecting the nodes, each representing the computation precedence order between two service-input tuples. For example, an edge $(\langle w_i, I_p\rangle, \langle w_j, I_q\rangle) \in E$ represents the precedence constraint indicating that a service-input tuple $\langle w_i, I_p\rangle$ must be processed before $\langle w_j, I_q\rangle$ is processed.
(iv) δ is a map which assigns a unique ordered pair of nodes to every edge.

In general, a relative composition request is a topologically sorted DAG $G\langle V, E\rangle$ such that a vertex $v \in V$, except root node, consists a service-input tuple (i.e., the message name belongs to the input set of the service). For instance, Fig. 6.2 and Fig. 6.3 are the examples of relative composition request, where Fig. 6.2 infers that flight availability must be checked by invoking the service $TA1$ followed by flight booking. Then, hotel availability must be checked by invoking the service $TA2$ followed by hotel booking. In contrast to Fig. 6.1, in Fig. 6.2, the respective requester services are specified for every message. This kind of composition request is suitable if a designer knows the available services in advance or she wants to make a specific composition request.

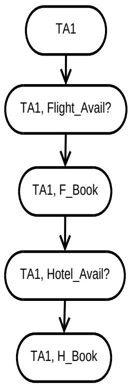


Figure 6.2: An Example of Relative Composition Request: Sequential Message Flow

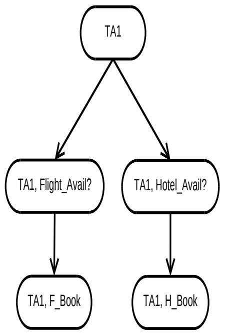


Figure 6.3: An Example of Relative Composition Request: Parallel Message Flow

A composition request graph G_{REQ} depicts a sequential message flow in a single branch and a parallel message flow in parallel branches. For example, Fig. 6.2 depicts a sequential message flow where a service TA_1 requests for flight booking and then, a service TA_2 requests for hotel booking. Fig. 6.3 depicts a parallel message flow where flight booking and hotel booking are requested in parallel.

Though RCG and a relative composition request graph (G_{REQ}) follow a similar structure, there are differences: (i) in a RCG, each node, except root, is successor of its parent node, whereas a G_{REQ} is free from this constraint, (ii) a RCG is automatically generated by applying recursive composition operation on a service-input tuple or service name or message name, whereas a G_{REQ} is manually created by a designer, and (iii) a pair of any two branches in a RCG are treated as non-deterministic choices, whereas a pair of any two branches in G_{REQ} are treated as parallel flows.

6.1.2 Web Service Composition Graph (WSCG)

Once a composition request is available, one has to explore all possible composition plans to realize the request with a set of available services $\mathcal{W}$. A Web service composition graph (WSCG) depicts all the possible composition plans for a given composition request. There are three entities that work as building blocks in a WSCG construction process: (i) a composition request G_{REQ} that has to be realized, (ii) a set of available services $\mathcal{W}$, and (iii) the recursive composition algebra (RCA). Given a composition request G_{REQ}, a WSCG is constructed by (i) applying recursive composition operator on each node in the given G_{REQ} that results RCGs with respect to the nodes, and (ii) replacing all the nodes in G_{REQ} with their respective RCGs. For instance, Fig. 6.4 shows the construction of the WSCG for the composition request depicted in Fig. 6.2. As is visible in Fig. 6.4, all the nodes in Fig. 6.2 are replaced by their respective RCGs. Like composition request graph, a WSCG is also a topologically sorted DAG and it preserves the structure of its originator composition request graph.

6.2 Composition Expression and Canonical Composition Expression

In this section, we describe the following two algebraic notions: *composition expression* and *canonical composition expression*.

6.2.1 Composition Expression

A composition expression is an algebraic representation of a G_{REQ} or RCG or WSCG. It is written with the help of three binary function symbols and service-input tuples. The three symbols are: '$\cdot$' for sequential, '$|$' for parallel, and '$+$' for non-deterministic choice. Given two service-message tuples $\langle w_i, I_p\rangle$ and $\langle w_j, I_q\rangle$

1. The expression '$\langle w_i, I_p\rangle \cdot \langle w_j, I_q\rangle$' represents the sequential composition that executes first $\langle w_i, I_p\rangle$ then $\langle w_j, I_q\rangle$.

2. The expression '$\langle w_i, I_p\rangle | \langle w_j, I_q\rangle$' represents the parallel composition that executes $\langle w_i, I_p\rangle$ and $\langle w_j, I_q\rangle$ in parallel.

3. The expression '$\langle w_i, I_p\rangle + \langle w_j, I_q\rangle$' represents the alternative composition that executes either $\langle w_i, I_p\rangle$ or $\langle w_j, I_q\rangle$ but not both.

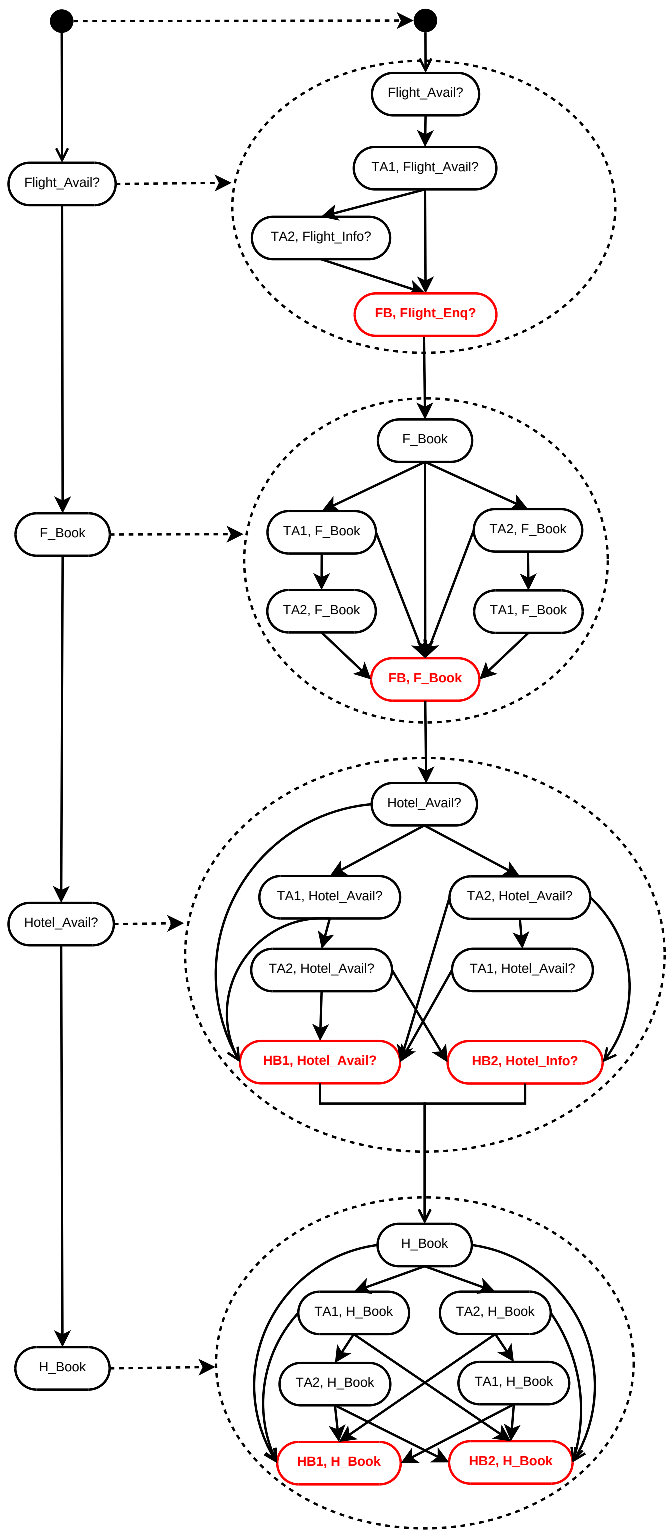


Figure 6.4: Web Service Composition Graph (WSCG) for a given service composition request depicted in Figure 6.1. A dotted circle, pointed by an arrowhead, consists of a RCG generated by applying recursive composition operation on the node at the arrow end.

All the functional symbols are of equal precedence. We use parentheses to induce the order in the expression.

Out of a given composition request graph G_{REQ} or RCG, the process of generating composition expression is provided in Algorithm 6.1. The algorithm accepts input as a DAG G and its source node s_0. In a DAG, a source node is a node with zero in-degree. Starting from the root node s_0, the algorithm processes every node of the graph in depth-first order, left to right sequence. Mainly, three variables are used in this algorithm: $CExp$, $CurrentNode$, and $Temp$.
(i) $CExp$ is a *String* type of variable that stores the composition expression. Initially, it is empty. Algorithm 6.1 generates the composition expression gradually and keeps updating the value of $CExp$. At the end of the algorithm, the complete composition expression becomes available in $CExp$.
(ii) $CurrentNode$ is a *String* type of variable that stores a node in-process. The variable $CurrentNode.ChildSet$ is a set that consists of the children of the $CurrentNode$.
(iii) $Temp$ is a *String* type of variable to store the temporary values.

The algorithm comprises a function Func($CExp, \langle w, I_p \rangle$) that accepts input as a composition expression $CExp$ and a node that consists of a service-message tuple $\langle w, I_p \rangle$. This function explores all the children of a given node and makes a subexpression by arranging them either with '+' symbol (if the given graph is a RCG) or with '|' symbol (if the given graph is a composition request). Then, the subexpression gets appended with the value of the current node by using '·' symbol. The generated subexpression, in $CExp$, replaces the $CurrentNode$. Further, the function considers the child nodes one by one and recursively calls itself by passing the arguments: $CExp$ and $\langle w, I_p \rangle$. For example, the composition expression for the RCG depicted in Fig. 3.1 is as follows:

$$\langle TA1, Hotel\_Avail? \rangle \cdot \Big(\langle HB1, Hotel\_Avail? \rangle + (\langle TA2, Hotel\_Avail? \rangle \cdot (\langle HB1, Hotel\_Avail? \rangle + \langle HB2, Hotel\_Info? \rangle))\Big)$$

Since a WSCG is made of a composition request and RCGs, its composition expression can also be computed by using Algorithm 6.1. Table 6.1 lists the various possible graph constructs in G_{REQ}/RCG/WSCG and their respective composition

Algorithm 6.1 CompExp(G, s_0)

```
    Input: A composition request graph or a RCG $G$ with source node $s_0$
    Output: CExp (composition expression for G)
 1: String $CExp \leftarrow \emptyset$
 2: let every node be represented in the format of service-input tuple ($\langle w, I_p \rangle$)
 3: function Func($CExp, \langle w, I_p \rangle$)
 4:     String $CurrentNode \leftarrow \langle w, I_p \rangle$
 5:     if $CurrentNode \cdot ChildSet \neq \emptyset$ then
 6:         int $Count \leftarrow 0$
 7:         for all $\langle w', I'_p \rangle \in CurrentNode \cdot ChildSet$ do
 8:             $Count = Count + 1$
 9:         end for
10:         if $Count = 1$ then
11:             if $CExp = \emptyset$ then
12:                 $CExp \leftarrow CurrentNode \cdot \langle w, I_p \rangle$
13:             else
14:                 replace $CurrentNode$ with $CurrentNode \cdot \langle w', I'_p \rangle$ in $CExp$
15:             end if
16:         end if
17:         if $Count > 1$ then
18:             String $Temp \leftarrow \emptyset$
19:             for all $\langle w', I'_p \rangle \in CurrentNode \cdot ChildSet$ do
20:                 if $Temp = \emptyset$ then
21:                     $Temp \leftarrow \langle w', I'_p \rangle$
22:                 else if $G =$ Composition Request then
23:                     $Temp \leftarrow Temp \mid \langle w', I'_p \rangle$
24:                 else if $G = RCG$ then
25:                     $Temp \leftarrow Temp + \langle w', I'_p \rangle$
26:                 end if
27:             end for
```

```
28:             $Temp \leftarrow CurrentNode \cdot Temp$
29:             if $CExp = \emptyset$ then
30:                 $CExp \leftarrow Temp$
31:             else
32:                 replace $CurrentNode$ with $Temp$ in $CExp$
33:             end if
34:         end if
35:         for all $\langle w', I'_p \rangle \in CurrentNode \cdot ChildSet$ do
36:             FUNC($CExp, \langle w', I'_p \rangle$)
37:         end for
38:     else if $CExp = \emptyset$ then
39:         $CExp \leftarrow CurrentNode$
40:     end if
41: end function
42: Return $CExp$
```

expressions.

6.2.2 Canonical Composition Expression

A canonical composition expression (canonical expression for short) is a special case of composition expression, written using service-input tuples for basic services and the three binary operators (mentioned earlier in this section). The reason why we consider only basic services in the generation of canonical expression is that in a Web service composition workflow, basic services are the ultimate service providers, and therefore, we consider *compositional equivalence* between two WSCGs based on: (i) candidate basic services, (ii) reachability of candidate basic services, (iii) relative position of candidate basic services in the given WSCGs, and (iv) compositional semantics of the WSCGs based on the candidate basic services. Canonical expression works as a base to compute the compositional equivalence between WSCGs. Given a RCG/WSCG, generating a canonical expression is a two-phase process:
(i) *Node reduction phase*. In this phase, one removes all the composite service nodes

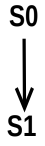
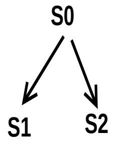
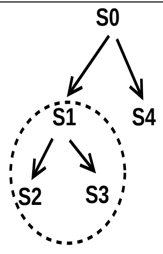
S.No.	Graph Construct	**Composition Expression** (if the graph construct is considered as a composition request)	**Composition Expression** (if the graph construct is considered as a RCG)	**Composition Expression** (if the graph construct is considered as a WSCG)
1	S0	S_0	S_0	S_0
2		$S_0 \cdot S_1$	$S_0 \cdot S_1$	$S_0 \cdot S_1$
3		$S_0 \cdot (S_1 \mid S_2)$	$S_0 \cdot (S_1 + S_2)$	$S_0 \cdot (S_1 \mid S_2)$
4		–	–	$S_0 \cdot \Big(\big(S_1 \cdot (S_2 + S_3)\big) \mid S_4\Big)$

Table 6.1: Composition Expressions for Various Graph Constructs. Serial no. 4 Depicts a Graph Construct that is a Special Case of WSCG, Where the Subgraph in Dotted Circle is Considered as a RCG

from the graph without affecting the relative arrangement of candidate basic services. (ii) *Composition expression generation phase*. In this phase, composition expression is generated for the graph resulted from the first phase. The generated composition expression is the desired canonical expression.

Let $\mathcal{C}$ be the representation for a canonical expression. $\mathcal{C}_\alpha$ represents a canonical expression corresponding to α (α could be a service-input tuple or a RCG or a G_{REQ}). Given a RCG, one can convert it into a canonical form by using Algorithm 6.2. The algorithm accepts input as a RCG G and its source node s_0. The canonical expression for a given RCG is constructed by using its end nodes and the alternative composition operator (+). For this, the algorithm computes all the end nodes by using the successor operator '$\succ$'. If a node does not have any successor node, then it is treated as an end node. Further, the algorithm arranges all the end nodes with '+' symbol. The reason why it uses only + symbol is that a pair of any two branches in a RCG are treated as non-deterministic choices. For example, the canonical composi-

Algorithm 6.2 CanCompExpForRCG(G, s_0)

```
    Input: RCG $G$ with source $s_0$
    Output: Canonical composition expression ($CCExp$)
 1: let every node be represented in the format of service-input tuple $\langle w, I_p\rangle$
 2: for all Node $\langle w, I_p\rangle \in G$ do
 3:     if $\succ (\langle w, I_p\rangle) = \emptyset$ then
 4:         $LeafNodeSet \leftarrow \langle w, I_p\rangle$
 5:     end if
 6: end for
 7: String $CCExp \leftarrow \emptyset$
 8: for all $\langle w, I_p\rangle \in LeafNodeSet$ do
 9:     if $CCExp = \emptyset$ then
10:         $CCExp \leftarrow \langle w, I_p\rangle$
11:     else
12:         $CCExp \leftarrow CCExp + \langle w, I_p\rangle$
13:     end if
14: end for
```

tion expression for the RCG depicted in Figure 3.1 (generated by applying recursive composition on $\langle TA_1, Hotel\_Avail?\rangle$) is given as follows:

$$\mathcal{C}_{\langle TA_1, Hotel\_Avail?\rangle} = \langle HB_1, Hotel\_Avail?\rangle + \langle HB_2, Hotel\_Info?\rangle \tag{6.1}$$

Equation 6.1 infers that the composition request $\langle TA_1, Hotel\_Avail?\rangle$ is ultimately fulfilled either by invoking $\langle HB_1, Hotel\_Avail?\rangle$ or by invoking $\langle HB_2, Hotel\_Info?\rangle$.

Given a composition request graph G_{REQ}, we convert it into a canonical form by using Algorithm 6.3. The algorithm accepts input as a graph G_{REQ} and its source node s_0. The algorithm computes the composition expression for the G_{REQ} and stores it in a variable $CCExp$. Then, for each term ($\langle w, I_p\rangle$ is considered as term) in $CCExp$, it constructs a RCG. Further, the canonical expression is computed for the constructed RCGs by calling Algorithm 6.2. Thus, one gets canonical expression for each term in $CCExp$. The computed canonical expressions replace their respective terms in $CCExp$. After all the replacements, $CCExp$ becomes the canonical expression for the given composition request graph.

In order to derive a canonical expression from a WSCG, its generator composition

Algorithm 6.3 CanCompExpForCompReq(G_{REQ}, s_0)

```
   Input: A composition request $G_{REQ}$ with source $s_0$
   Output: Canonical composition expression ($CCExp$)
1: let every node be represented in the format of service-input tuple ($\langle w, I_p\rangle$)
2: $CCExp \leftarrow$ CompExp($G, \langle w, I_p\rangle$)
3: for all $\langle w, I_p\rangle \in CCExp$ do
4:     $G' \leftarrow$ RecComp($\langle w, I_p\rangle, \mathcal{W}$)
5:     $\mathcal{C}_{\langle w, I_p\rangle} \leftarrow$ CanCompExpForRCG($G', \langle w, I_p\rangle$)
6:     replace $\langle w, I_p\rangle$ with $\mathcal{C}_{\langle w, I_p\rangle}$ in $CCExp$
7: end for
```

request graph (G_{REQ}) is also required. By passing the request graph to Algorithm 6.3 as an argument, the canonical expression for the WSCG is computed. For example, expression 6.2 is the canonical expression for the graph shown in Figure 6.4.

$$\langle FB, Flight\_Enq?\rangle \cdot \langle FB, F\_Book\rangle \cdot \Big(\langle HB_1, Hotel\_Avail?\rangle + \langle HB_2, Hotel\_Info?\rangle\Big) \cdot (\langle HB_1, H\_Book\rangle + \langle HB_2, H\_Book\rangle) \quad (6.2)$$

6.3 Web Services Compositional Equivalence Verification Technique

In this section, we describe a formal technique for verifying compositional equivalence between WSCGs. This technique is based on the concepts of canonical expression (described in Section 6.2) and subsumption computing. Fig. 6.5 presents a

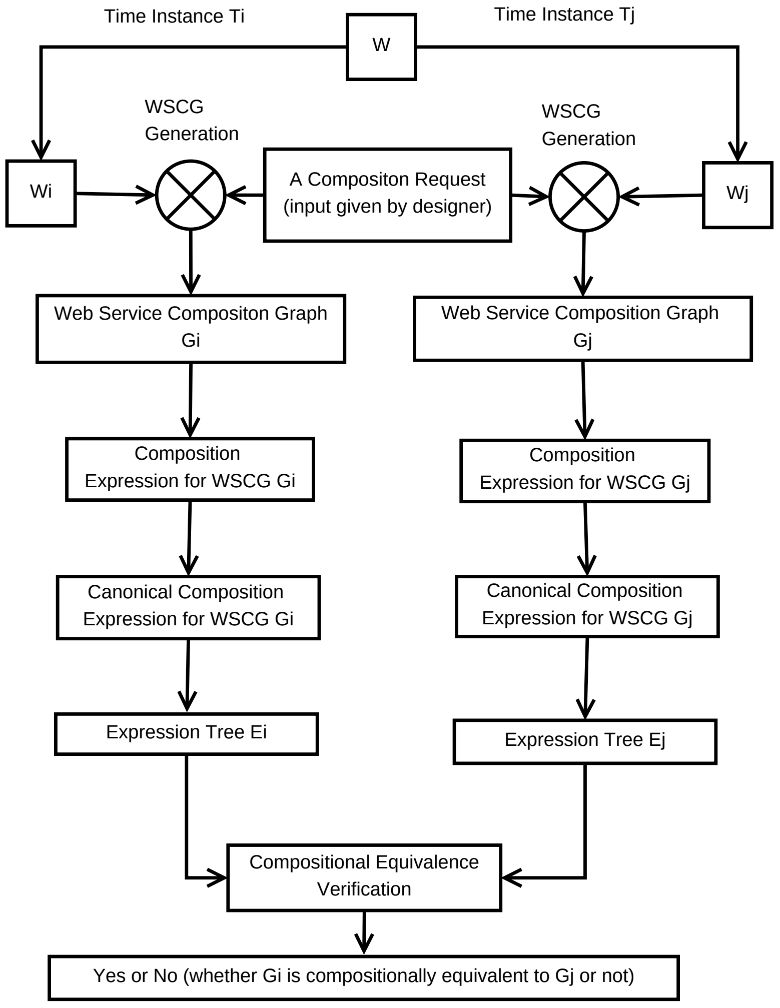

Time Instance Ti
W
Time Instance Tj
WSCG Generation
WSCG Generation
Wi
A Compositon Request (input given by designer)
Wj
Web Service Compositon Graph Gi
Web Service Composition Graph Gj
Composition Expression for WSCG Gi
Composition Expression for WSCG Gj
Canonical Composition Expression for WSCG Gi
Canonical Composition Expression for WSCG Gj
Expression Tree Ei
Expression Tree Ej
Compositional Equivalence Verification
Yes or No (whether Gi is compositionally equivalent to Gj or not)

Figure 6.5: Block Diagram of the Compositional Equivalence Verification Process

block diagram of the verification process presented in this chapter. Suppose we are provided with a composition request graph (say G_{REQ}) and a set of Web services (say $\mathcal{W}$). Depending upon the requirement, G_{REQ} may be an absolute request or a relative request. As we have already mentioned that $\mathcal{W}$ is a dynamic set, a service might become unavailable or a new service can join the set at any time instance. For our verification process, we consider two instances of $\mathcal{W}$: (i) $\mathcal{W}_i$ at time instance t_i and (ii) $\mathcal{W}_j$ at time instance t_j (where $t_j > t_i$). Further, we assume that $\mathcal{W}_j \subset \mathcal{W}_i$, inferring that at time t_j some services become unavailable in comparison to the available ones at time t_i. Upon receiving G_{REQ}, $\mathcal{W}_i$, and $\mathcal{W}_j$, the proposed verification process generates two WSCGs (say G_i and G_j corresponding to $\mathcal{W}_i$ and $\mathcal{W}_j$) out of the given request G_{REQ} by applying recursive composition operation and Algorithm 6.1. The WSCGs G_i and G_j depict exhaustive possibilities to realize the G_{REQ} with the help of $\mathcal{W}_i$ and $\mathcal{W}_j$ respectively. Now, our primary interest is in knowing that whether G_j can functionally substitute G_i or not. The verification process considers G_i as a pattern graph and G_j as a subject graph and verifies the pattern graph against the subject graph. Since a subgraph matching problem is a NP-Complete problem [202], in our approach, we shift this problem to a problem of semantic matching between two binary expression trees. Given a WSCG, getting the binary expression tree is a three-step process: (i) generation of the composition expression for the given WSCG; (ii) generation of the canonical expression out of the generated composition expression; (iii) construction of the binary expression tree out of the generated canonical expression.

In order to form the binary expression tree for a given canonical composition expression, we transform it into a postfix expression, and then, from the postfix expression, we generate the expression tree. The conversion of an expression into a postfix expression and the conversion of a postfix expression into an expression tree are well-known processes. Hence, we are not describing algorithms for these two processes. Subsumption between binary expression trees is defined as follows:

Definition 6.3.1 (Subsumption between Binary Expression Trees)**.** Let E_i and E_j be two binary expression trees, then, E_i subsumes E_j if and only if 'true' truth value of E_i always makes E_j true.

This definition is also applicable for canonical expressions. Further, Algorithm 6.4 presents a technique for subsumption computing between two binary expression trees. Let E_i and E_j be the two binary expression trees for the pattern and subject canonical expressions respectively. Let $Root_i$ and $Root_j$ be the root nodes of E_i and E_j respectively. $FLAG$ is a Boolean variable that is initially set to $TRUE$ and becomes $FALSE$ if an anomaly is encountered in the subsumption computing process. After the complete execution of the algorithm, if $FLAG$ remains $TRUE$, then it infers that E_j subsumes E_i. Then, we repeat the process by reversing the input order. That is, we check whether E_i subsumes E_j or not. If this is so, we can conclude that G_i and G_j are equivalent in terms of the functionalities they provide.

Algorithm 6.4 SubsumptionComputing(E_i, E_j)

```
    Input: Binary Expression Trees $E_i$ and $E_j$
    Output: Decides whether $E_j$ subsumes $E_i$ or not
 1: $R_i \leftarrow Root(E_i)$
 2: $R_j \leftarrow Root(E_j)$
 3: Boolean $FLAG \leftarrow TRUE$
 4: function SubsumCheck($R_i, R_j$)
 5:     if $R_i \neq \emptyset$ & $R_j \neq \emptyset$ then
 6:         if $R_i = R_j$ then
 7:             if $R_i.LeftChild = R_j.LeftChild$ then
 8:                 if $R_i.RightChild = R_j.RightChild$ then
 9:                     RecProceed($R_i, R_j$)
10:                 else if $R_i.RightChild =$ '+' then
11:                     $R_i \leftarrow R_i.RightChild$
12:                     if $R_i.RightChild = R_j.RightChild$ then
13:                         RecProceed($R_i, R_j$)
14:                     end if
15:                 else
16:                     $FLAG \leftarrow FALSE$
17:                     Quit          //Negative result
18:                 end if
19:             else if $R_i.RightChild = R_j.RightChild$ then
20:                 if $R_i.LeftChild =$ '+' then
21:                     $R_i \leftarrow R_i.LeftChild$
22:                     if $R_i.LeftChild = R_j.LeftChild$ then
23:                         RecProceed($R_i, R_j$)
24:                     end if
25:                 else
26:                     $FLAG \leftarrow FALSE$
27:                     Quit          //Negative result
28:                 end if
29:             end if
30:         end if
```

```
31:     else if $R_i =$ '$+$' & $R_i.LeftChild = R_j$ then
32:         $R_i \leftarrow R_i.LeftChild$
33:         SubsumCheck($R_i, R_j$)
34:     else if $R_i =$ '$+$' & $R_i.RightChild = R_j$ then
35:         $R_i \leftarrow R_i.RightChild$
36:         SubsumCheck($R_i, R_j$)
37:     else
38:         $FLAG \leftarrow FALSE$
39:         Quit        //Invalid input
40:     end if
41: end function
42: function RecProceed($R_i, R_j$)
43:     $R_i \leftarrow R_i.LeftChild$
44:     $R_j \leftarrow R_j.LeftChild$
45:     SubsumCheck($R_i, R_j$)
46:     $R_i \leftarrow R_i.RightChild$
47:     $R_j \leftarrow R_j.RightChild$
48:     SubsumCheck($R_i, R_j$)
49: end function
50: if ( then$FLAG = TRUE$)
51:     $E_j$ subsumes $E_i$
52: else
53:     $E_j$ does not subsume $E_i$
54: end if
```

Formally, on the basis of subsumption computation between two canonical expressions, the compositional equivalence between two WSCGs is defined as follows:

Definition 6.3.2 (Compositional Equivalence). Let G_i and G_j be two WSCGs and $CCExp_i$ and $CCExp_j$ be two canonical expressions for G_i and G_j respectively. Let $CCExp_i \sqsubset CCExp_j$ represents that $CCExp_j$ subsumes $CCExp_i$. Then, G_i and G_j are compositionally equivalent if and if only if $CCExp_i \sqsubset CCExp_j$ and $CCExp_j \sqsubset CCExp_i$.

Instead of providing a single composition request graph G_{REQ} and two instances

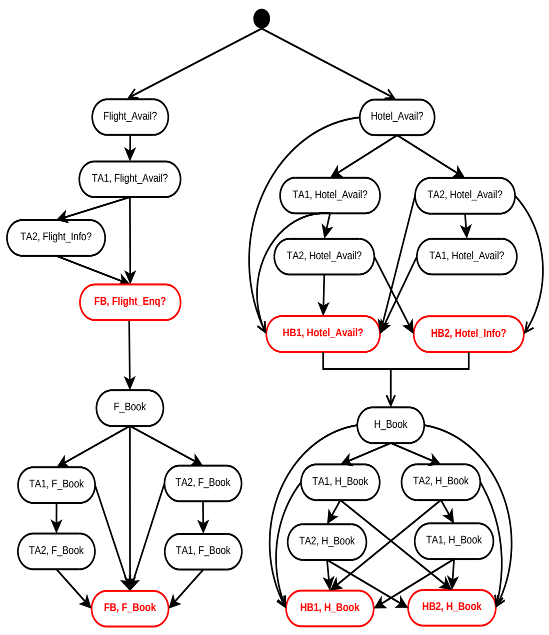


Figure 6.6: Pattern WSCG G_i

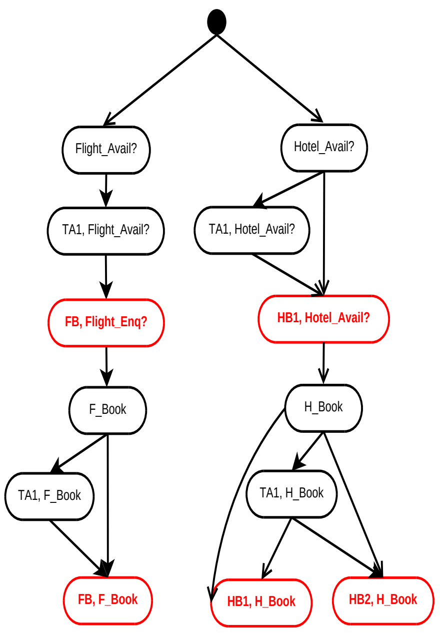


Figure 6.7: Subject WSCG G_j

of $\mathcal{W}$ (namely $\mathcal{W}_i$ and $\mathcal{W}_j$) as inputs, two different request graphs with their respective sets of Web services ($\mathcal{W}_i$ and $\mathcal{W}_j$) can also be provided as inputs to the verification process. Moreover, the proposed algorithms are also applicable for the following two variations of our consideration: (i) some services become unavailable and some new services join at the time instance t_j (ii) some new services join and no service become unavailable at the time instance t_j.

Example 6.3.1. Suppose we are given with a composition request, namely G_{REQ}, depicted in Fig. 6.3 and a set of Web services $\mathcal{W}$ (described in Chapter 3. We assume that at a time instance t_i all the services in $\mathcal{W}$ are available and at a time instance t_j service TA_2 is unavailable. Let $\mathcal{W}_j$ be a set of available Web services at t_j {i.e. $\mathcal{W}_j = (\mathcal{W} - TA_2)$}. Now, the verification process generates two WSCGs: G_i (see Figure 6.6) corresponding to t_i and $\mathcal{W}$ and G_j (see Figure 6.7) corresponding to t_j and W_j. In the verification process, G_i works as a pattern WSCG and G_j works as a subject WSCG. The composition expressions corresponding to WSCGs G_i and G_j

are given by expression 6.3 and 6.4 respectively.

$$\langle Flight\_Avail?\rangle\cdot(\cdots)\cdot\langle F\_Book\rangle\cdot(\cdots)\ \Big|\langle Hotel\_Avail?\rangle\cdot(\cdots)\cdot\langle H\_Book\rangle\cdot(\cdots) \tag{6.3}$$

$$\begin{aligned}&\langle Flight\_Avail?\rangle\cdot\langle TA_1, Flight\_Avail?\rangle\cdot\langle FB, Flight\_Enq?\rangle\cdot\\ &\quad\langle F\_Book\rangle\cdot\Big((\langle TA_1, F\_Book\rangle\cdot\langle FB, F\_Book\rangle)+\langle FB, F\_Book\rangle\Big)\ \Big|\\ &\langle Hotel\_Avail?\rangle\cdot((\langle TA_1, Hotel\_Avail?\rangle\cdot\langle HB_1, Hotel\_Avail?\rangle)+\langle HB_1, Hotel\_Avail?\rangle)\\ &\quad\langle H\_Book\rangle\cdot\Big((\langle TA_1, H\_Book\rangle\cdot(\langle HB_1, H\_Book\rangle+\langle HB_2, H\_Book\rangle))+\\ &\qquad(\langle HB_1, H\_Book\rangle)+(\langle HB_2, H\_Book\rangle)\Big)\end{aligned} \tag{6.4}$$

Further, the canonical composition expressions derived from the expressions 6.3 and 6.4 are given by expressions 6.5 and 6.6 respectively.

$$\begin{aligned}&\Big(\langle FB, Flight\_Enq?\rangle\cdot\langle FB, F\_Book?\rangle\Big)\ \Big|\\ &\qquad\Big((\langle HB_1, Hotel\_Avail?\rangle+\langle HB_2, Hotel\_Info?\rangle)\cdot\\ &\qquad\qquad(\langle HB_1, H\_Book\rangle+\langle HB_2, H\_Book\rangle)\Big)\end{aligned} \tag{6.5}$$

$$\begin{aligned}&\Big(\langle FB, Flight\_Enq?\rangle\cdot\langle FB, F\_Book\rangle\Big)\ \Big|\\ &\qquad\Big(\langle HB_1, Hotel\_Avail?\rangle\cdot(\langle HB_1, H\_Book\rangle+\langle HB_2, H\_Book\rangle)\Big)\end{aligned} \tag{6.6}$$

The binary expression trees for the expressions 6.5 and 6.6 are given by Fig. 6.8 and 6.9 respectively. After computing the subsumption by using Algorithm 6.4, we get assured that G_j subsumes G_i, thereby, we can conclude that unavailability of the Web service TA_2 does not affect the realization of the composition request G_{REQ}.

6.3.1 Computational Complexity

In the verification technique, various processes are involved such as RCG generation, WSCG generation, etc. All of these processes contribute to complexity. However, we are concerned with only subsumption computation between two binary expression trees as it plays pivotal role in the verification process. In our proposition, a binary expression tree consists of operators and service-input tuples. There

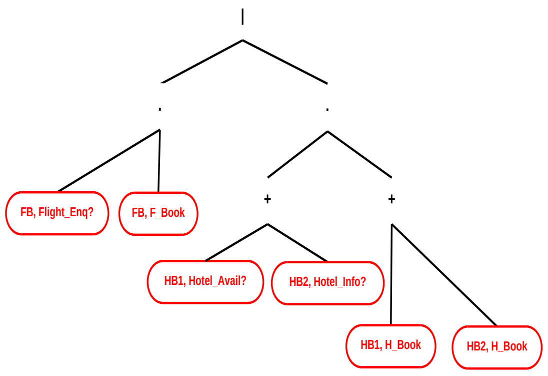


Figure 6.8: Binary Expression Tree for The Canonical Composition Expression of Pattern WSCG

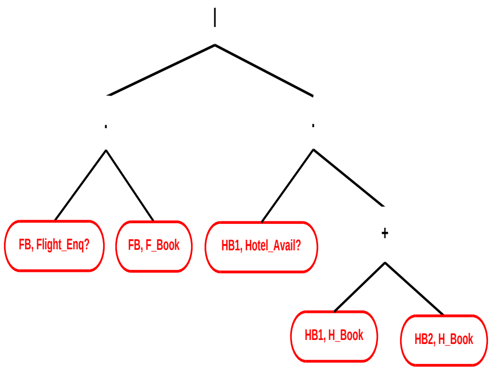


Figure 6.9: Binary Expression Tree for the Canonical Composition Expression of Subject WSCG

are three operators that can appear in the tree: '$\cdot$','$|$', and '$+$'. Let E_1 and E_2 be two binary expression trees with n_1 and n_2 number of nodes respectively. In the worst case, subsumption computation process between E_1 and E_2 resembles the binary subtree matching problem. Therefore, the complexity becomes $O(n_1 \times n_2)$. The non-deterministic choice operator ('$+$') reduces the number of comparisons between two trees by half. Therefore, in the average case, the complexity becomes $O(\log n_1 \times \log n_2)$ and in the best case, the complexity becomes $O(\log n_2)$.

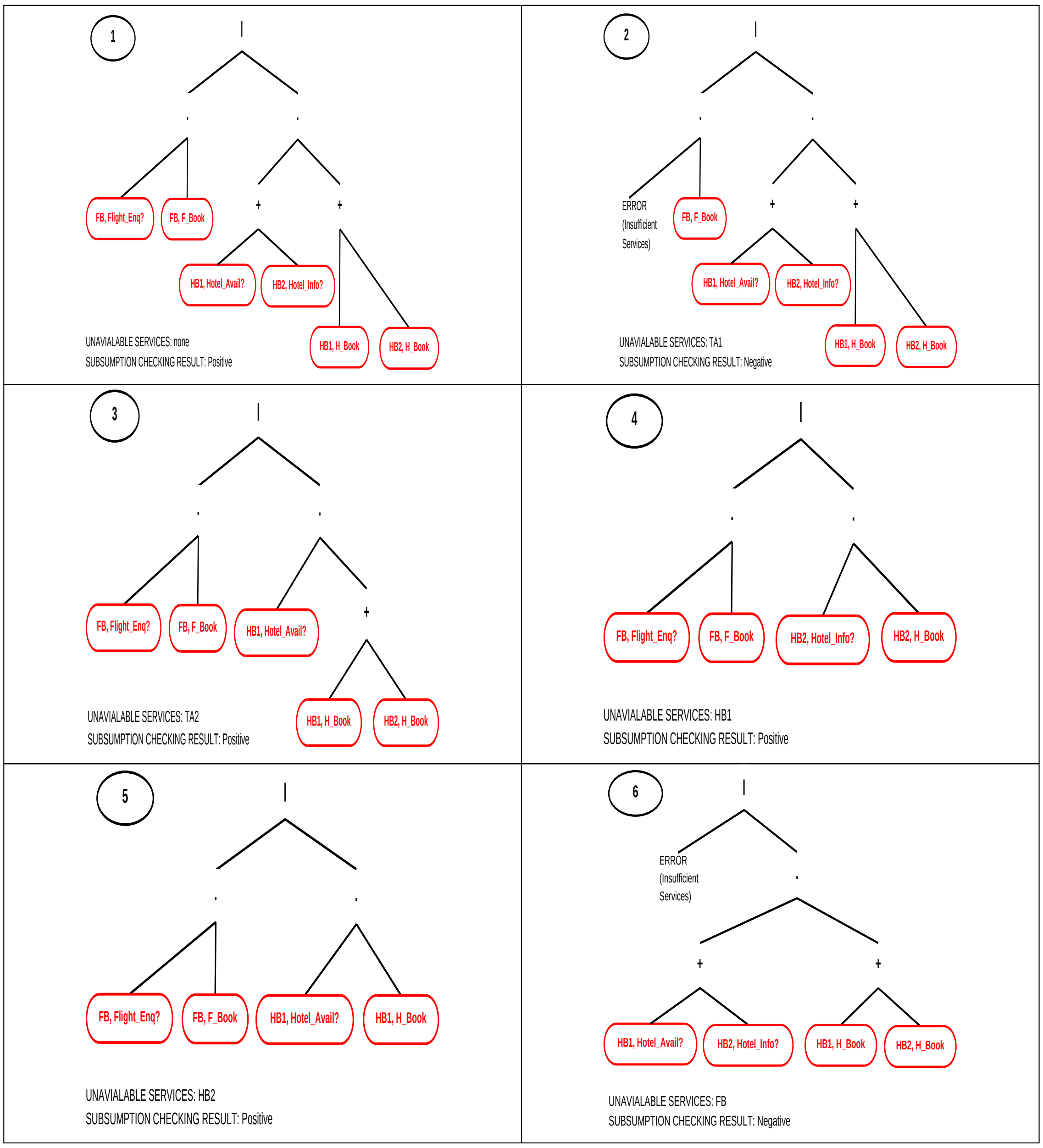
6.4 Experimental Evaluation

6.4.1 Web Services Repository

Due to the unavailability of the standard test web service repository in the verification domain, the experimental evaluation of the proposed technique has been performed on the service repository created by ourselves. This repository consists of ten services as mentioned in Chapter 3. All the services are deployed independently and heterogeneously. We did not consider more services as it would be very difficult to demonstrate all the possible cases in this chapter. Further, we consider up to 150 services for the scalability study of this verification technique.

6.4.2 Experiments and Observations

Experiments are carried out in lab conditions, that is, services and repositories are deployed on the systems connected to a Local Area Network (LAN) and they are using local IP addresses. The composition request, depicted in Fig. 6.1, is provided as an argument. The supplied request is written in the dot language (an input language for the GraphViz tool). The composition request is analyzed and the respective WSCG (see Fig. 6.7) is produced. We call the WSCG in Fig. 6.7 as *standard one*. In our implementation, we use GraphViz tool for graph realization and visualization. Five out of ten available services in the repository fall in the category of relevant services: TA_1, TA_2, HB_1, HB_2, and FB. Intentionally, we make the relevant services unavailable (first individually and then collectively) and generate the WSCGs with available services. These constructed WSCGs show differences from the standard one as some of the relevant services are unavailable and we call them as changed ones. Further, we compute composition expressions and canonical composition expressions out of the constructed WSCGs. After computing the canonical expressions, we generate binary expression trees out of them. Table 6.2 consists of ten binary expression trees (in short, trees). The trees are numbered 1 to 10 for the sake of reference. The first tree is generated based on the standard WSCG and we call it as *standard binary expression tree* (standard tree, in short). Other trees (numbered 2–10) are generated from the changed WSCGs. Further, each tree is computed for subsumption against the standard one (first tree) and list of unavailable services and result of subsumption computation are written therewith. In a tree, *insufficient services* condition occurs if a composite service does not find basic services to compose with. For instance, the second diagram in Table 6.2 infers that second tree is generated by considering that $TA1$ is unavailable and subsumption result is *negative* as it does not subsume the standard one. In the similar way, subsumption computing is done for all other trees too. At last, from the experiment, we deduce the following results: (i) at a given time, the unavailability of anyone of { TA_1, FB, (TA_2, HB_1), (HB_1, HB_2) } makes realization impossible for the given composition request (Fig. 6.1), and (ii) at a given time, the unavailability of anyone of { TA_2, HB_1, HB_2, (TA_2, HB_2) } does not affect the realization of the given composition request (Fig. 6.1).


1
FB, Flight_Enq?
FB, F_Book
HB1, Hotel_Avail?
HB2, Hotel_Info?
HB1, H_Book
HB2, H_Book
UNAVIALABLE SERVICES: none
SUBSUMPTION CHECKING RESULT: Positive
2
ERROR (Insufficient Services)
FB, F_Book
HB1, Hotel_Avail?
HB2, Hotel_Info?
HB1, H_Book
HB2, H_Book
UNAVIALABLE SERVICES: TA1
SUBSUMPTION CHECKING RESULT: Negative
3
FB, Flight_Enq?
FB, F_Book
HB1, Hotel_Avail?
HB1, H_Book
HB2, H_Book
UNAVIALABLE SERVICES: TA2
SUBSUMPTION CHECKING RESULT: Positive
4
FB, Flight_Enq?
FB, F_Book
HB2, Hotel_Info?
HB2, H_Book
UNAVIALABLE SERVICES: HB1
SUBSUMPTION CHECKING RESULT: Positive
5
FB, Flight_Enq?
FB, F_Book
HB1, Hotel_Avail?
HB1, H_Book
UNAVIALABLE SERVICES: HB2
SUBSUMPTION CHECKING RESULT: Positive
6
ERROR (Insufficient Services)
HB1, Hotel_Avail?
HB2, Hotel_Info?
HB1, H_Book
HB2, H_Book
UNAVIALABLE SERVICES: FB
SUBSUMPTION CHECKING RESULT: Negative

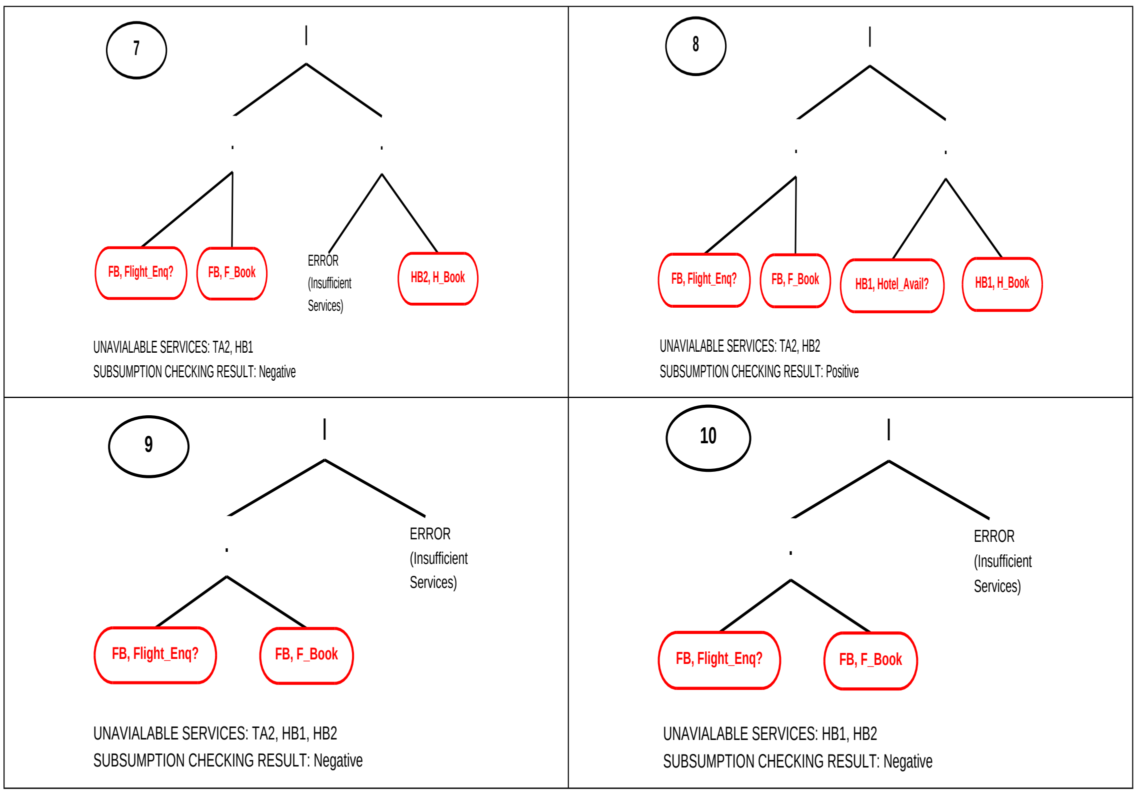

7
FB, Flight_Enq?
FB, F_Book
ERROR (Insufficient Services)
HB2, H_Book
UNAVIALABLE SERVICES: TA2, HB1
SUBSUMPTION CHECKING RESULT: Negative
8
FB, Flight_Enq?
FB, F_Book
HB1, Hotel_Avail?
HB1, H_Book
UNAVIALABLE SERVICES: TA2, HB2
SUBSUMPTION CHECKING RESULT: Positive
9
ERROR (Insufficient Services)
FB, Flight_Enq?
FB, F_Book
UNAVIALABLE SERVICES: TA2, HB1, HB2
SUBSUMPTION CHECKING RESULT: Negative
10
ERROR (Insufficient Services)
FB, Flight_Enq?
FB, F_Book
UNAVIALABLE SERVICES: HB1, HB2
SUBSUMPTION CHECKING RESULT: Negative

Table 6.2: Binary Expression Trees Generated in the Process of Validation of the Presented Compositional Equivalence Verification Technique

Though we have used a small set of Web services (consisting of ten services) for the demonstration purpose, our proposed verification process is scalable to a large number of services. Given a set of available services and a composition request, the verification process consists of three stages: (i) construction of WSCG, (ii) finding canonical composition expression, and (iii) subsumption computation. The construction of WSCG is the highest time and space consuming process, among the said three stages. Therefore, we show experimental evidence on the time taken by WSCG construction process. The experiments are performed for three composition requests as given in the Fig. 6.10, 6.11, and 6.12. For experimentation, we used a personal computer with an Intel Core i5-3230M 2.60 GHz CPU, 2 GB RAM, 500 GB HDD, and Ubuntu 14.04 64-bit operating system. Table 6.3 presents the time consumption (measured in seconds) in the process of WSCGs generation for the three absolute composition requests shown in Fig. 6.10, 6.11, and 6.12. We considered

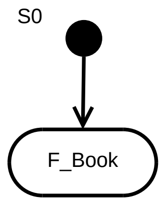


Figure 6.10: Composition Request 1: A Composition Request with a Single Non-Empty Node

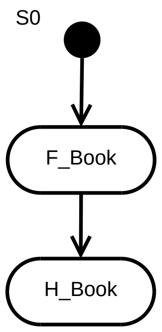


Figure 6.11: Composition Request 2: A Composition Request with Two Non-Empty Nodes

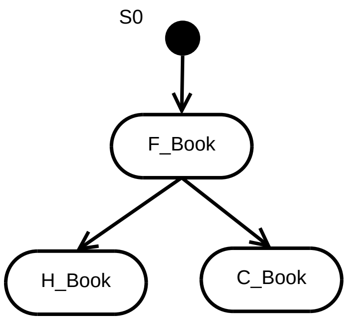


Figure 6.12: Composition Request 3: A Composition Request with Three Non-Empty Nodes

composition requests in such a way that they represent different graph constructs (single node, sequential, branching), number of nodes in them increase gradually, and realization of them is possible with the available services mentioned in Section 3.2. Based on the data provided in Table 6.3, time consumption plots for the WSCG generation are shown in Fig. 6.13. For each composition request, we varied the number of candidate services from 30 to 150 with steps of 30 (e.g., 30, 60, 90, 120, 150). In order to increase the number of candidate services, we created new services by replicating the candidate services available in the set W. The plots, in Fig. 6.13, infer that: (i) If equal number of services are available for each composition request graph, then the time consumption is proportional to the number of nodes in the graph. It happens due to the fact that the WSCG generation process generates a RCG with respect to every node in the given request graph. (ii) Increment in the number of

S.No.	Comp. Request	# Candidate Services	Time (in seconds)
1	Comp. Request 1	30	70
2	Comp. Request 1	60	185
3	Comp. Request 1	90	312
4	Comp. Request 1	120	750
5	Comp. Request 1	150	1478
1	Comp. Request 2	30	143
2	Comp. Request 2	60	316
3	Comp. Request 2	90	824
4	Comp. Request 2	120	1941
5	Comp. Request 2	150	4229
1	Comp. Request 3	30	173
2	Comp. Request 3	60	445
3	Comp. Request 3	90	1005
4	Comp. Request 3	120	2197
5	Comp. Request 3	150	4590

Table 6.3: Time Consumption in the Process of WSCGs Generation for the Composition Requests Shown in Fig. 6.10, 6.11, and 6.12

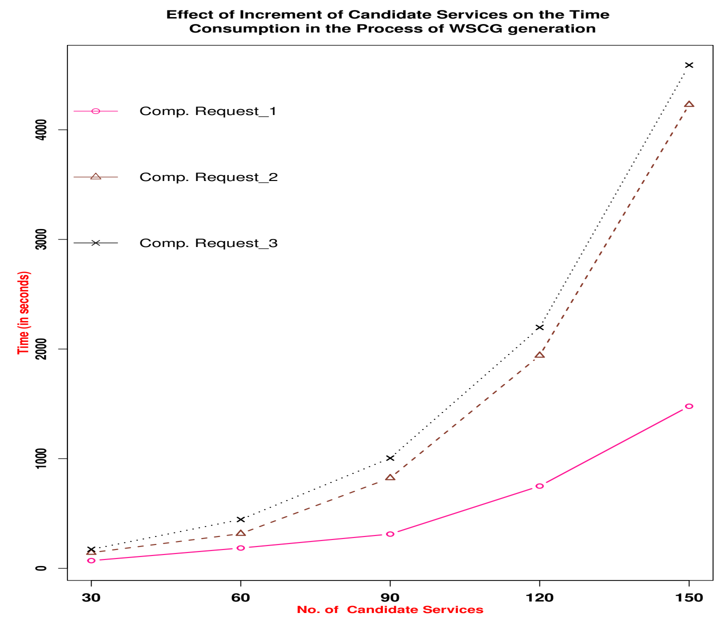


Figure 6.13: Effect of Increment of Candidate Services on the Time Consumption in the Process of WSCG Generation

candidate services increases the time consumption exponentially. The reason behind this exponential time consumption are: (i) recursive composition operation has to recursively search into the available Web service space, and (ii) generation of the RCG/WSCG without using a graph heuristic or optimization technique. In our future work, we would like to consider the techniques that might reduce the search space, thus reducing RCG/WSCG generation time.

6.5 Related Work

6.5.1 Composition Request Graph, Compositional Equivalence, and Binary Expression Tree

The composition request graph, used in this chapter, is inspired from the workflow modeling language *Yet Another Workflow Language (YAWL)* [203]. In fact, the set of control-flow constructs used in the request graph is a subset of the control-flow constructs set available in YAWL. In our future work, we also would like to consider other control-flow constructs in the construction of composition request graph. Another work that used a graphical representation for the composition request is [200]. We differ from [200] in that their focus was only on the sequential composition model, whereas our proposed composition request graph captures both sequential and parallel composition models.

In the context of Web services, the concept of compositional equivalence was mentioned explicitly in [204]. We differ from [204] mainly in two aspects: (1) we investigate the compositional equivalence between WSCGs, whereas [204] investigated the compositional equivalence between services, and (2) we determine the compositional equivalence based on the fulfillment of a given composition request, whereas [204] determined the compositional equivalence based on the behavior of services.

The notion of binary expression tree for a composite Web service, where leaves represent the atomic services and intermediate leaves represent workflow constructors, was proposed in [205]. In our proposal, we also adopt the same concept. However, we differ from [205]: (i) in [205], author used the binary tree to realize a composite plan for a given composition request, whereas we are generating the binary tree from an already constructed composition plan, and (ii) unlike to [205], we use the binary tree for verifying compositional equivalence between WSCGs.

6.5.2 Web Service Substitutability Verification

To the best of our knowledge, we do not find a work directly focused on the verification of compositional equivalence between Web Service Composition Graphs (WSCGs). However, our work is closely related with the context-dependent substitutability verification of Web services. Therefore, we compare our work with context-dependent substitutability verification approaches.

The compatibility between two services ensures that they interact properly without any error. Klai et al. [25] used *Symbolic Observation Graph* (SOG) to determine compatibility between two services. The SOG nodes are aggregation of a set of local states which are connected with non-observed actions, and they do not contain service names. This makes difficult to monitor the impact of change in the set of available services. Klai and Ochi [206] also used the SOG for checking compatibility of Web services behavior. Corrales et al. [207] presented a substitutability verification technique that accepts two BPEL or two WSCL documents as input. In our opinion, BPEL and WSCL are used to represent the choreography plan, and availability or unavailability of a service has nothing to do with BPEL or WSCL matchmaking. Moreover, the graphical representation of a BPEL or WSCL document in [207] cannot serve our purpose of capturing all possible composition plans out of a set of available services.

Unlike to previously discussed compatibility-dependent notions, Pathak et al. [111] presented the concept of environment based substitutability notions: environment dependent and environment independent. Their logical formulation of the solution is based on the well-studied notion of "quotienting". However, Pathak et al. did not consider the messages being exchanged by the services which is considered in our approach. Santhanam et al. [208] proposed a variation of the approach in [111]. However, they differ from [111] in their consideration of non-functional attributes for substitutability. Stahl et al. [209] proposed an environment-based solution for service substitutability. According to [209], there are three notions of substitutability that can exist between two services, namely *accordance*, *equivalent*, and *deprecation*. Their proposal for service substitution is based on service automata and operating guidelines. We differ from [209] fundamentally in that we investigate substitutability based on the compositional equivalence, whereas their [209] investigation is based on behavioral equivalence. Kuang et al. [210] modeled a Web service and context using π-calculus. Further, they introduced a notion of behavioral equivalence by considering the context of a service. Runtime checking of service substitutability based on satisfiability of LTL specifications was proposed in [123]. However, decidability of their approach is very complex.

Taher et al [211] proposed an interface-matching based solution for the problem of service substitutability. In this approach, every candidate service belongs to some pre-specified Web service community. In case, if a service becomes unavailable from a community, other services from the community could be employed. Liang et al. [212] also followed a similar approach of classifying Web services using their descriptions. Motahari-Nezhad et al. [186] advocated that the process of checking service equivalence should not be done merely on the basis of interface, its business protocol also must be considered. In order to enhance interface-matching experience, they provided two types of interface-matching algorithms: *depth-based interface matching* and *iterative reference-based interface matching*. Another interface-based similarity measurement between services was studied in [213]. They computed substitutability on the basis of various interface matching techniques and semantic and structural similarity metrics. We differ from [186, 211, 212] and [213] in that we compute the service equivalence on the basis of fulfillment of a given composition request instead of interface.

In general, in contrast to other mentioned proposals on service behavior and substitutability (i) we define algebra-based evolution of a Web Service Composition Graph (WSCG) from the composition request graph, and (ii) we propose an algebra-based compositional equivalence verification technique, that is not NP-Complete. The overview of major existing approaches for Web service substitutability and differences with our proposed approach are shown in Table 6.4.

6.6 Summary

We modeled the service composition on the basis of RCA and further defined Web service composition graph (WSCG) based on it. A WSCG is the graphical composition plan of how a set of Web services compose with each other to realize a

Paper	Web service behavior modeling	Environment modeling		Substitution finding technique	Computational complexity	Implementation details
		Context-independent	Context-dependent			
Bordeaux et al. [110]	Labeled Transition System	√	√	Based on notion of compatibility	–	–
Liu et al. [117]	Process algebra	√	–	Based on process algebra constructs	–	–
Taher et al. [211]	Abstract and concrete Web services	√	–	Based on community of Web services	–	√
Grigori et al. [214]	Web service conversation language	√	√	Based on the concept of subgraph isomorphism	$O(m^n \times n)$, where m and n are total number of vertices in the input graph and in the graph to be compared, respectively	√
Pathak et al. [111]	Labeled Transition System	√	√	Based on Mu-calculus	$O(\|S\|^{nd} \times 2^{\|\phi\| \times B})$, where S is the set of states in LTS and B is branching factor	–
Bonchi et al. [204]	Petri net	–	√	Based on the concept of saturated bisimulation	–	–
Stahl et al. [209]	Open net and service automata	√	–	Based on operating guidelines	–	–
Motihari-Nezhad et al. [186]	As a triplet of data types, set of operations, and set of messages	√	–	Based on protocol-aware service matching algorithms	–	√
Bersani et al. [123]	Labeled Transition System	√	√	Based on Linear Temporal Logic	–	√
Kuang et al. [116]	Process algebra	√	√	Based on the concept of bisimulation	–	√
Li et al. [24]	Colored Petri net	√	–	Based on Petri net compatibility analysis	–	√
Tibermacine et al. [213]	As per WSDL documentation	√	–	Based on services WSDL interface	–	√
Klai et al. [25]	open workflow net	√	–	Based on the notion of compatibility	–	√
Our approach	Service behavior graph (SBG)	$--$	√	Based on subsumption computing between algebraic expressions	$\{O(n_1 \times n_2\}$, where n_1 and n_2 are sizes of the SBG_1 and SBG_2	√

Table 6.4: An Overview of Major Existing Approaches for Web Service Substitutability and Differences with Our Approach

given composition request. Given a dynamic set of available services and the desired service behavior in the form of a composition request graph, our proposition verifies the compositional equivalence between two WSCGs formed before and after the structural change caused by the change in the set of available services. The novelty of the presented approach is that it verifies the compositional equivalence between two WSCGs based on the subsumption computing between two algebraic expressions. This kind of verification avoids the need of complex subgraph matching process between the composition graphs and reduces the computational complexity associated with the verification process to a greater extent.

Chapter 7

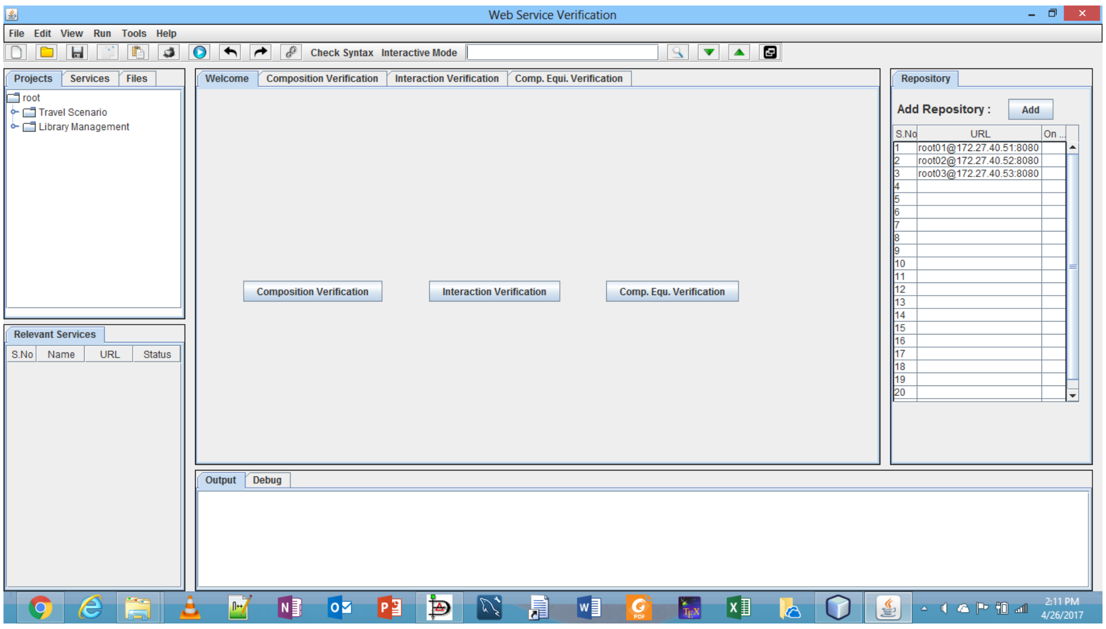
An Integrated Tool for Web Services Verification

In this chapter, we present the description of an integrated tool for the verification of Web services composition, interaction, and compositional equivalence. This tool is based on the verification techniques presented in Chapter 4, Chapter 5, and Chapter 6.

7.1 An Integrated Tool for Web Services Verification

There are three divisions in the tool: composition verification, interaction verification, and compositional equivalence verification between WSCGs. The core technique used for modeling is *Recursive Composition Algebra* (RCA), proposed in Chapter 3. With the help of RCA, three types of models are generated: *Web service set partition graph (WSSP)*, *recursive composition interaction graph* (RCIG), and *Web service composition grpah* (WSCG). WSSP is employed in the composition verification process, RCIG is employed in the interaction verification process, and WSCG is employed in the compositional equivalence verification process. For the composition verification, a user has to specify a set of Web services. Once a set of Web services is specified, the tool automatically verifies behavioral equivalence between every pair of two services and finds all possible deadlock conditions. For the interaction verification, a user has to specify the requirements in RCSL (described in Chapter 5). If the given specification is not satisfied, a counter trace will be generated, otherwise, a witness example will be generated. For the compositional equivalence verification, a user has to provide two WSCGs. Then, the tool verifies whether they are compositionally equivalent or not.

Fig. 7.1 shows the welcome window of the tool, where a user can select a desired module to proceed with the verification process. This tool has been implemented in Java and is based on the technologies such as XML, SOAP, and WSDL. Further, it requires a repository of Web services, and the repository can be provided either as a database of WSDL documents or by Internet addresses (URLs).



S.No	URL	On ...
1	root01@172.27.40.51:8080	
2	root02@172.27.40.52:8080	
3	root03@172.27.40.53:8080	
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		

Figure 7.1: An Integrated Tool for Web Services Verification: Welcome Window

7.1.1 Composition Verification

In this subsection, we describe a prototype implementation of the Web service composition verification technique presented in Chapter 4. Fig. 7.2 depicts a high-level architecture of the implementation that consists of five modules: RCG Generator, WSSP Graph Generator, Composition Planner, Goal Sets Generator, and Verifier and Analyzer. Apart from the five modules, the implementation also consists of an online Web service repository that is a collection of WSDL documents of available services. The said five modules are detailed as follows.

1. **RCG Generator**. Given a service name or message name or service-input tuple, this module generates the corresponding recursive compostion graph (RCG) by using the recursive composition operation.

2. **WSSP Graph Generator**. This module generates the WSSP graph for a given service or service-input tuple by taking the help of *RCG Generator* module and Algorithm 4.1.

3. **Composition Planner**. This module explores the WSSP graph for a given requirement specification and finds all the possible ways to realize the specification. For this, it takes the help of *WSSP Graph Generator* module.

4. **Goal Sets Generator**. Given a set of Web services, this module generates the goal sets by using the WSSP graph and the definition of goal sets. Further, the generated goal sets are provided as input to the *Verifier and Analyzer* module.

5. **Verifier and Analyzer**. Given a set of Web services, this module analyzes the behavioral equivalence between the services and the possible deadlock scenarios by using the concept of goal sets and Algorithm 4.3.

Fig. 7.3 shows a screenshot of the user interface (UI) used for composition verification. This UI shows a list of all available Web service repositories, where a user selects the repositories for verification and analysis. Once the repositories are selected, the behavioral equivalence between services and the possible deadlock scenarios appear in the UI's output pane automatically.

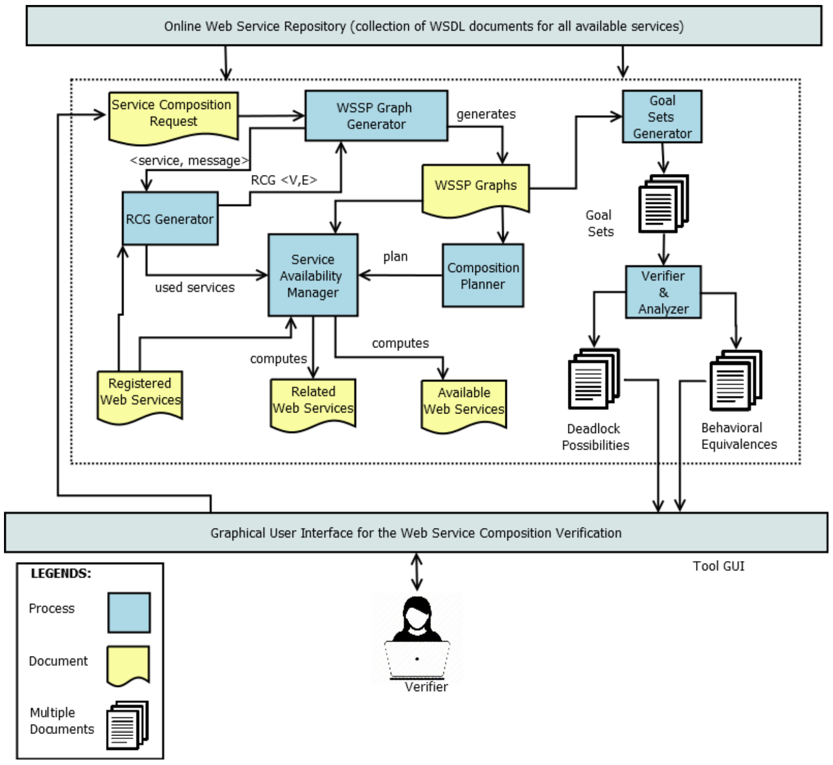
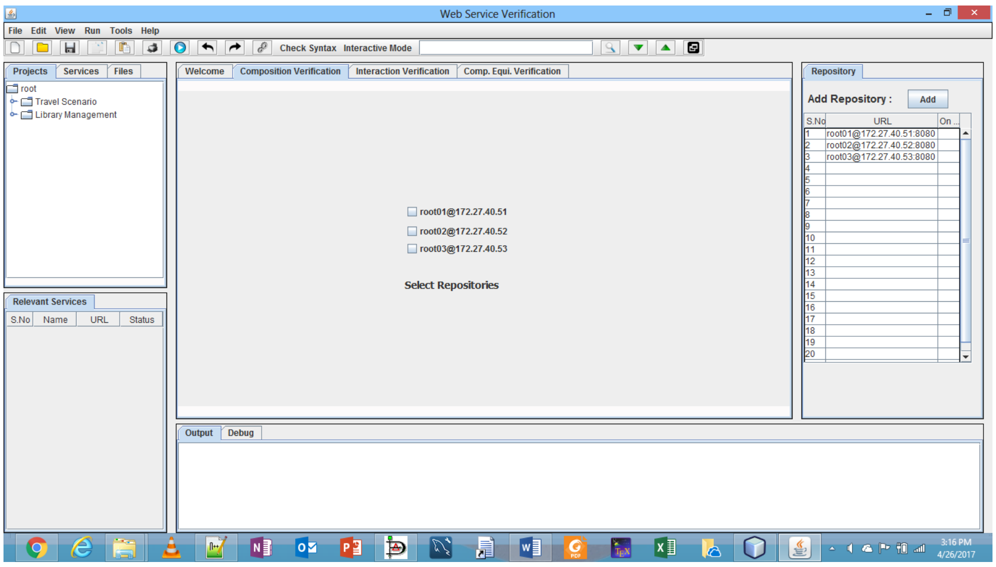


Figure 7.2: High-Level Architecture of Implementation of Web Service Composition Verification



S.No	URL	On ...
1	root01@172.27.40.51:8080	
2	root02@172.27.40.52:8080	
3	root03@172.27.40.53:8080	
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		

S.No	Name	URL	Status

Figure 7.3: Composition Verification User Interface

7.1.2 Interaction Verification

In this subsection, we describe a prototype implementation of the verification technique described in Chapter 5 for verifying the specifications written in the RCSL against the set of available Web services. Fig. 7.4 shows the high-level architecture of the prototype system that consists of four modules: specification formula parsing, intermediate form conversion, RCIG and trace generator, and semantical interpretation.

1. **Specification Formula Parsing**: This module receives a requirement specification formula from the verifier and processes it using Algorithm 5.4. Syntax checking is performed at first. It makes an abstract syntax tree (AST) out of the given formula. The generated AST is decomposed into its constituent subtrees until the unit-level-subtrees are achieved. The unit-level-subtrees are provided to the module *semantical interpretation*.

2. **Intermediate Form Conversion**: This module receives the specification formula and discovers the set of relevant services from the available ones. Then, it retrieves their WSDL documents and makes a duplicate (local) copy of WSDL documents and modifies them by adding two tags: sequential and parallel. The modified WSDL documents work as the intermediate representation and are provided to the module *RCIG and trace generator* for further processing.

3. **RCIG and Trace Generator**: This module receives the set of modified WSDL documents along with an input (a service-input tuple or a message name or a service name). The input is provided by Algorithm 5.3. Once an input and the set of modified WSDL documents are available, it forms a RCIG using Algorithm 5.1. Further, it optimizes the generated RCIG by removing redundant subtraces and supplies the traces to the module *semantical interpretation*.

4. **Semantical Interpretation**: This module verifies whether a given trace interprets the semantics of a given subtree (subformula). If the trace satisfies the formula, then the trace would be considered as a witness example. Otherwise, the trace would be considered as a counter example.

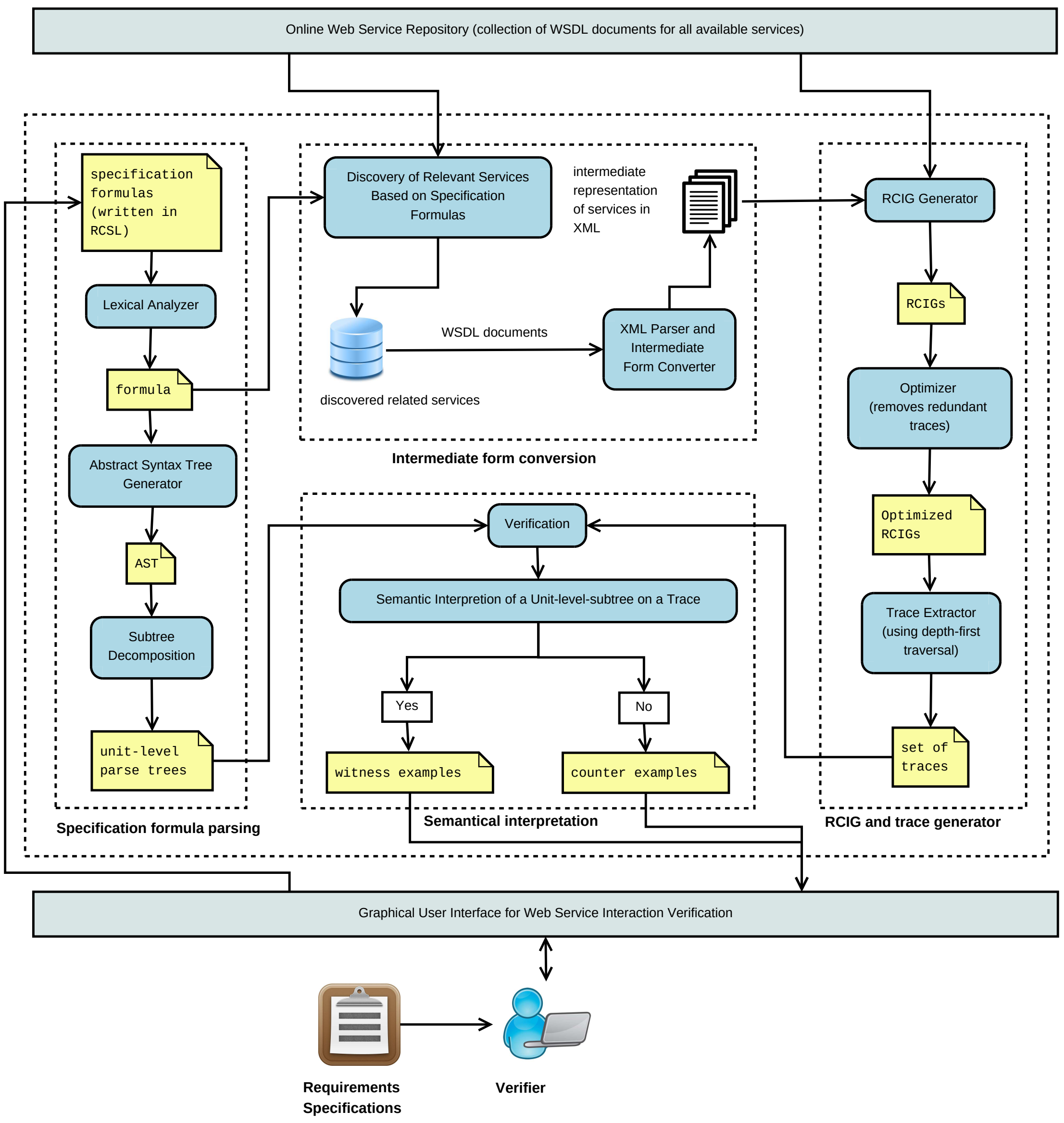
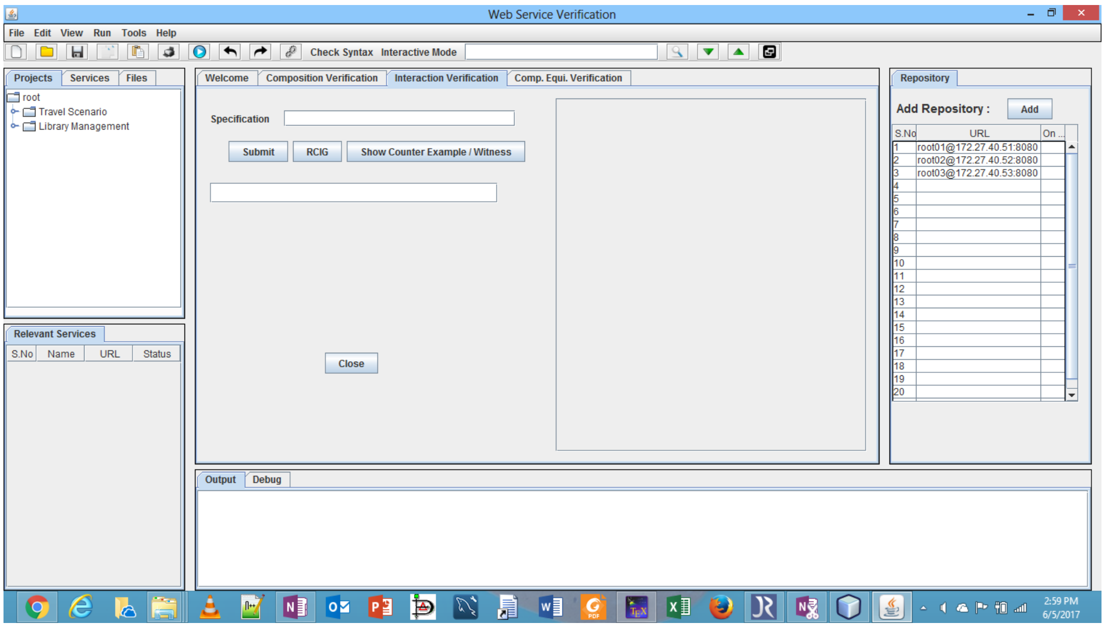

Online Web Service Repository (collection of WSDL documents for all available services)
specification formulas (written in RCSL)
Lexical Analyzer
formula
Abstract Syntax Tree Generator
AST
Subtree Decomposition
unit-level parse trees
Specification formula parsing
Discovery of Relevant Services Based on Specification Formulas
intermediate representation of services in XML
WSDL documents
discovered related services
XML Parser and Intermediate Form Converter
Intermediate form conversion
Verification
Semantic Interpretion of a Unit-level-subtree on a Trace
Yes
No
witness examples
counter examples
Semantical interpretation
RCIG Generator
RCIGs
Optimizer (removes redundant traces)
Optimized RCIGs
Trace Extractor (using depth-first traversal)
set of traces
RCIG and trace generator
Graphical User Interface for Web Service Interaction Verification
Requirements Specifications
Verifier

Figure 7.4: High-Level Architecture of Implementation of Web Service Interaction Verification

In addition to automation, this implementation also supports dynamic reconfiguration of services. A Web service verification framework where the verifier has to decide the participant services in advance (with or before specifying the requirement) does not support dynamic reconfiguration of services. However, in our approach, a specification formula can be written with or without explicitly mentioning the participant service names. If service names are not explicit, then the relevant services are discovered based on the atoms in the formula.



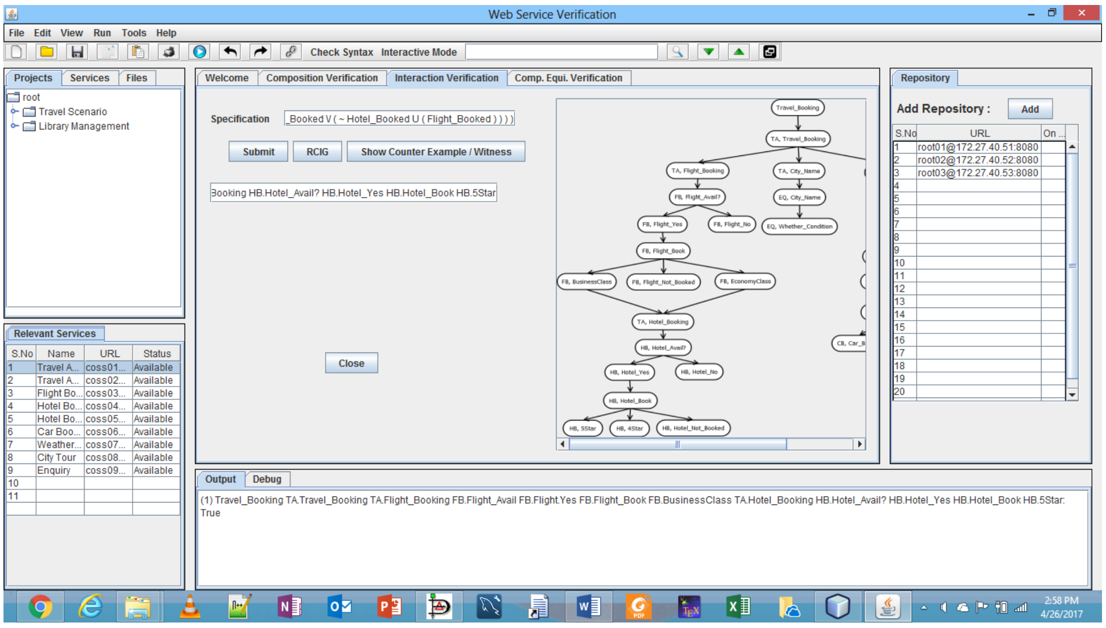
S.No	URL	On ...
1	root01@172.27.40.51:8080	
2	root02@172.27.40.52:8080	
3	root03@172.27.40.53:8080	
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		

Figure 7.5: Graphical User Interface for Verification of the Requirement Specifications Written in RCSL

Initially, if required, the verifier has to specify only the addresses of local or global service repositories. During verification, our implementation discovers the relevant services and checks the current availability of services each time when the verification query is submitted and generates the RCIG models accordingly.

Also, the implementation features a graphical user interface (see Fig. 7.5) that supports a wide range of features such as editing and tracking of the modeled system, writing the requirements specifications in RCSL, adding new services in the repository by specifying their addresses, checking syntax, and starting verification.

Fig. 7.6 shows the GUI screen with three specifications (say, ϕ_1, ϕ_2, and ϕ_3) and a RCIG generated for ϕ_1. Fig. 7.7 shows a witness example for the first specification ($\phi_1 = A((Flight\_Yes \wedge Flight\_Book) \rightarrow (Flight\_Booked \vee (\neg Hotel\_Booked \; \cup \; (Flight\_Booked))))$).



Specification _Booked V (~ Hotel_Booked U (Flight_Booked))))

Booking HB.Hotel_Avail? HB.Hotel_Yes HB.Hotel_Book HB.5Star

Relevant Services

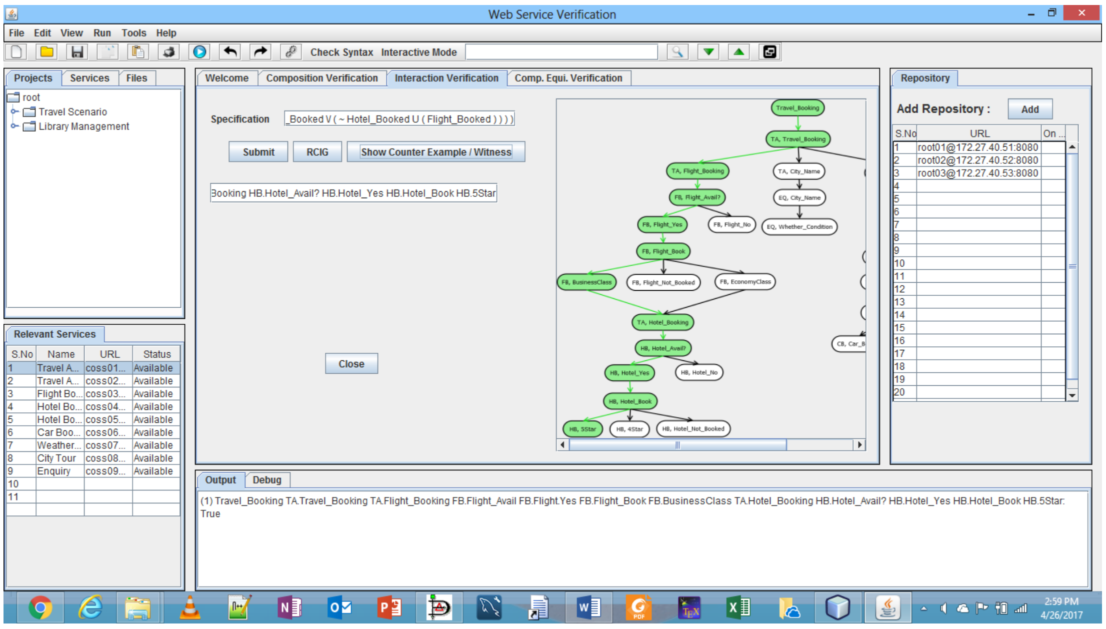
S.No	Name	URL	Status
1	Travel A...	coss01...	Available
2	Travel A...	coss02...	Available
3	Flight Bo...	coss03...	Available
4	Hotel Bo...	coss04...	Available
5	Hotel Bo...	coss05...	Available
6	Car Boo...	coss06...	Available
7	Weather...	coss07...	Available
8	City Tour	coss08...	Available
9	Enquiry	coss09...	Available
10			
11			

Add Repository :

S.No	URL	On ...
1	root01@172.27.40.51:8080	
2	root02@172.27.40.52:8080	
3	root03@172.27.40.53:8080	
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		

```
(1) Travel_Booking TA.Travel_Booking TA.Flight_Booking FB.Flight_Avail FB.Flight.Yes FB.Flight_Book FB.BusinessClass TA.Hotel_Booking HB.Hotel_Avail? HB.Hotel_Yes HB.Hotel_Book HB.5Star:
True
```

Figure 7.6: Verification of Specification Formulas and RCIG



Specification _Booked V (~ Hotel_Booked U (Flight_Booked))))

Booking HB.Hotel_Avail? HB.Hotel_Yes HB.Hotel_Book HB.5Star

Relevant Services

S.No	Name	URL	Status
1	Travel A...	coss01...	Available
2	Travel A...	coss02...	Available
3	Flight Bo...	coss03...	Available
4	Hotel Bo...	coss04...	Available
5	Hotel Bo...	coss05...	Available
6	Car Boo...	coss06...	Available
7	Weather...	coss07...	Available
8	City Tour	coss08...	Available
9	Enquiry	coss09...	Available
10			
11			

Add Repository :

S.No	URL	On ...
1	root01@172.27.40.51:8080	
2	root02@172.27.40.52:8080	
3	root03@172.27.40.53:8080	
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		

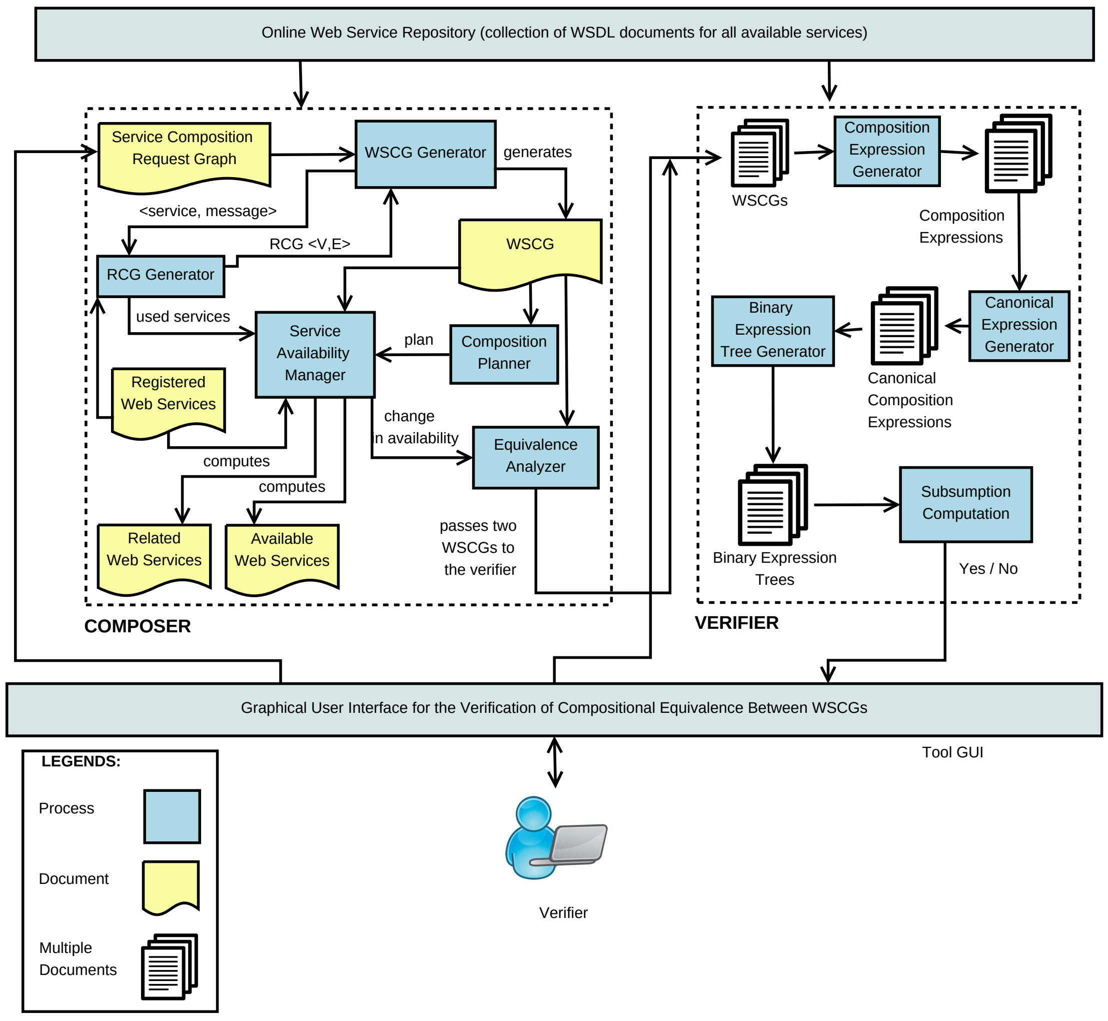
```
(1) Travel_Booking TA.Travel_Booking TA.Flight_Booking FB.Flight_Avail FB.Flight.Yes FB.Flight_Book FB.BusinessClass TA.Hotel_Booking HB.Hotel_Avail? HB.Hotel_Yes HB.Hotel_Book HB.5Star:
True
```

Figure 7.7: Witness Example

Furthermore, the described verification framework is fully capable to verify other systems that behave similarly to the Web service system (such as multi-agent system), provided that their system description is in the desired XML format and the requirement specifications are written in RCSL.

7.1.3 Compositional Equivalence Verification

In this subsection, we describe the implementation for verifying the compositional equivalence between two WSCGs, depicted in Chapter 6. Fig. 7.8 depicts the high-level architecture that consists of two modules: Composer and Verifier. A user or designer is free to access either of the two modules directly and independently by using a graphical user interface (GUI). Both modules are detailed as follows:



Online Web Service Repository (collection of WSDL documents for all available services)
Service Composition Request Graph
WSCG Generator
generates
<service, message>
RCG <V,E>
RCG Generator
WSCG
used services
Service Availability Manager
plan
Composition Planner
Registered Web Services
change in availability
Equivalence Analyzer
computes
computes
Related Web Services
Available Web Services
passes two WSCGs to the verifier
COMPOSER
WSCGs
Composition Expression Generator
Composition Expressions
Canonical Expression Generator
Binary Expression Tree Generator
Canonical Composition Expressions
Subsumption Computation
Binary Expression Trees
Yes / No
VERIFIER
Graphical User Interface for the Verification of Compositional Equivalence Between WSCGs
Tool GUI
LEGENDS:
Process
Document
Multiple Documents
Verifier

Figure 7.8: High-Level Architecture of Compositional Equivalence Verification Tool

(1) Composer. A user provides two input arguments to this module: an address of a Web service repository and a desired composition request (modeled as G_{REQ}). The composer consists of the following submodules:

1. **RCG Generator**. It accepts the input in the form of a service-input tuple. Based on the given input, it identifies the relevant services from the repository and generates the respective RCG. Further, it forwards the list of utilized services to the service availability manager.

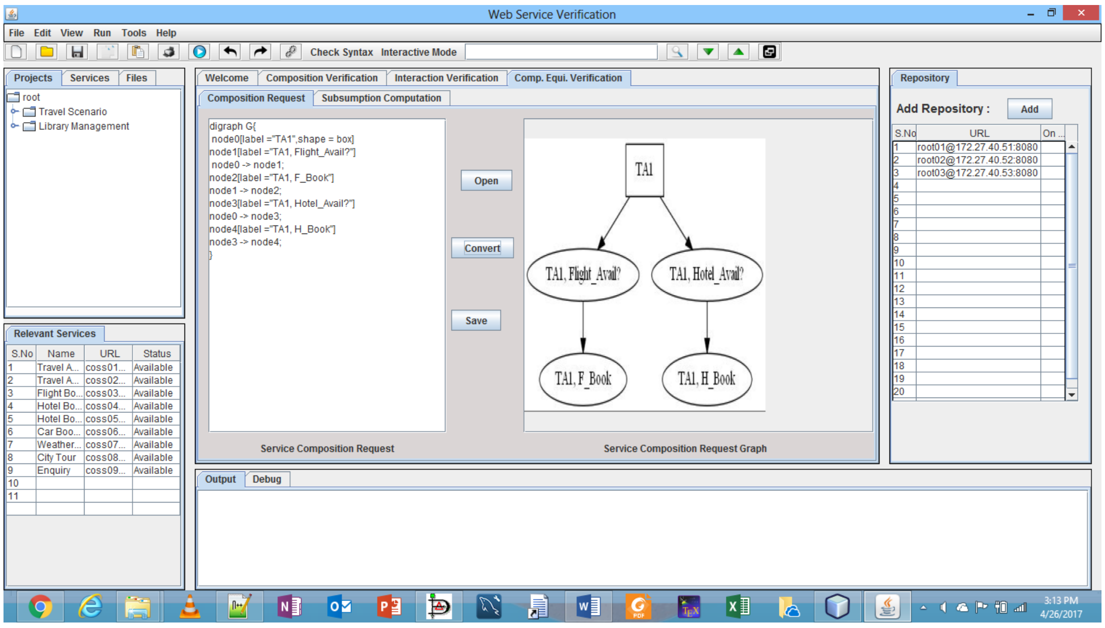
2. **WSCG Generator**. It analyzes the given composition request and forms respective WSCG with the help of submodule *RCG generator*.

3. **Service Availability Manager**. It runs continuously in background and keeps track of related services for a WSCG and their availability. In order to support dynamic availability, we intentionally add new services and remove available ones on random basis. Once it learns that a related service has become unavailable or a new related service is added to the repository, it passes the description of unavailable services to the substitutability analyzer for further processing.

4. **Composition Planner**. It explores all possible composition patterns for a given composition request and selects a suitable one from them based on the requirements. Further, it passes the selected plan to the submodule service availability manager.

5. **Substitutability Analyzer**. It accepts two inputs: a reference WSCG and a set of unavailable and newly added services.

Fig. 7.9 depicts a GUI, where the composer module is functioning. There are two modes available for composer: automatic and manual. In automatic mode, it composes and substitutes (if necessary) without human intervention. In manual mode, a user/designer selects a composition plan from a set of available plans and selects a substitute when required.



```
digraph G{
 node0[label ="TA1",shape = box]
node1[label ="TA1, Flight_Avail?"]
 node0 -> node1;
node2[label ="TA1, F_Book"]
node1 -> node2;
node3[label ="TA1, Hotel_Avail?"]
node0 -> node3;
node4[label ="TA1, H_Book"]
node3 -> node4;
}
```

S.No	Name	URL	Status
1	Travel A...	coss01...	Available
2	Travel A...	coss02...	Available
3	Flight Bo...	coss03...	Available
4	Hotel Bo...	coss04...	Available
5	Hotel Bo...	coss05...	Available
6	Car Boo...	coss06...	Available
7	Weather...	coss07...	Available
8	City Tour	coss08...	Available
9	Enquiry	coss09...	Available
10			
11			

S.No	URL	On ...
1	root01@172.27.40.51:8080	
2	root02@172.27.40.52:8080	
3	root03@172.27.40.53:8080	
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		

Figure 7.9: A Composition Request Graph (Written in Dot Language) as an Input to the Composer Module

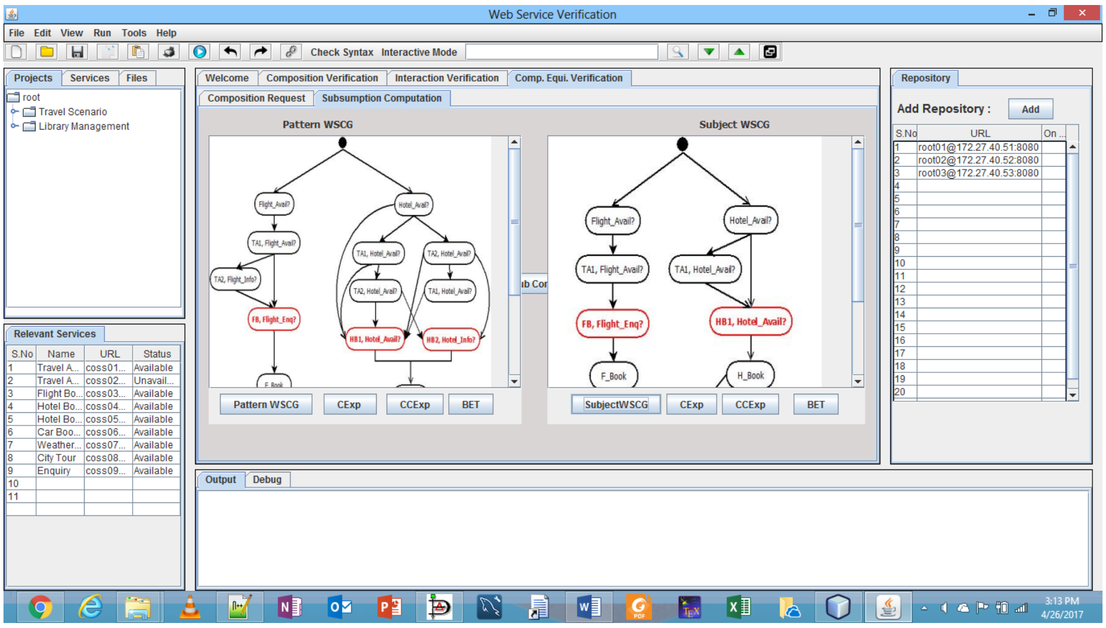
(2) Verifier. This module requires two input arguments; both of the input arguments are WSCGs. Further, it verifies whether the semantics of a provided WSCG could be achieved by another one. In order to accomplish this task, the verifier comprises the following three submodules:

1. **Composition Expression Generator.** It generates the composition expressions out of given WSCGs using Algorithm 6.1 and forwards them to the submodule *canonical composition expression generator*.

2. **Canonical Composition Expression Generator.** It works on a composition expression for a given WSCG and transforms it into the canonical form.

3. **Subsumption Computation.** It constructs binary expression trees for the given canonical composition expressions and verifies whether the semantics of a binary expression tree is subsumed by another one.

Figure 7.10 depicts a GUI, where the verifier module is functioning.



Relevant Services

S.No	Name	URL	Status
1	Travel A...	coss01...	Available
2	Travel A...	coss02...	Unavail...
3	Flight Bo...	coss03...	Available
4	Hotel Bo...	coss04...	Available
5	Hotel Bo...	coss05...	Available
6	Car Boo...	coss06...	Available
7	Weather...	coss07...	Available
8	City Tour	coss08...	Available
9	Enquiry	coss09...	Available
10			
11			

S.No	URL	On ...
1	root01@172.27.40.51:8080	
2	root02@172.27.40.52:8080	
3	root03@172.27.40.53:8080	
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		

Figure 7.10: Two WSCGs (WSCG1: When All Services are Available, and WSCG2: When TA2 is Unavailable) as Inputs for Compositional Equivalence Verification

7.2 Summary

This chapter discussed the implementation details of the composition verification technique (described in Chapter 4), the interaction verification technique (described in Chapter 5), and the compositional equivalence verification technique (described in Chapter 6) as an integrated tool for Web services verification.

Chapter 8

Application of RCA in Network Security: Multistage Attack Modeling Using RCA

Detection of multistage Cyber attacks in an enterprise network is of growing interest to the security community. Existing threat modeling techniques, such as attack graphs and attack trees, do not facilitate reasoning about the causal relationship between network vulnerabilities. Consequently, the area of multistage attack needs more exploration. In this chapter, we describe a multistage attack modeling technique based on RCA (described in Chapter 3). For a given vulnerable network configuration, the algebra generates a recursive composition graph (RCG) that depicts all possible multistage attack scenarios. The prime advantage of the RCG is that it is free from cycles, therefore, does not require the computation intensive cycle detection algorithms. Further, the canonical sets obtained from the RCG classifies network vulnerabilities into five classes: (i) isolated, (ii) strict igniter (entry point), (iii) strict terminator (dead end) (iv) overlapping, and (v) mutually exclusive. These classes (logical inferences) provide better insight to the logical correlation among existing vulnerabilities in a given network and hence in prioritizing vulnerability remediation activities accordingly.

8.1 Running Example

We consider a test network whose topology is shown in Figure 8.1. There are six machines located within two subnets. A Web server and a mail server are located inside the demilitarized zone (DMZ), and are separated from the local network by a Tri-homed DMZ firewall. The firewall has a strong set of connectivity-limiting policies (as shown in Table 8.1) to prevent an adversary from gaining remote access to the internal hosts. All service requests (coming from outside of the network) are fulfilled through the machines in the DMZ. In Table 8.1, *ALL* specifies that a source host may access all services running over the destination host and *NONE* specifies that a source host is prevented from having access to any service running over the destination host.

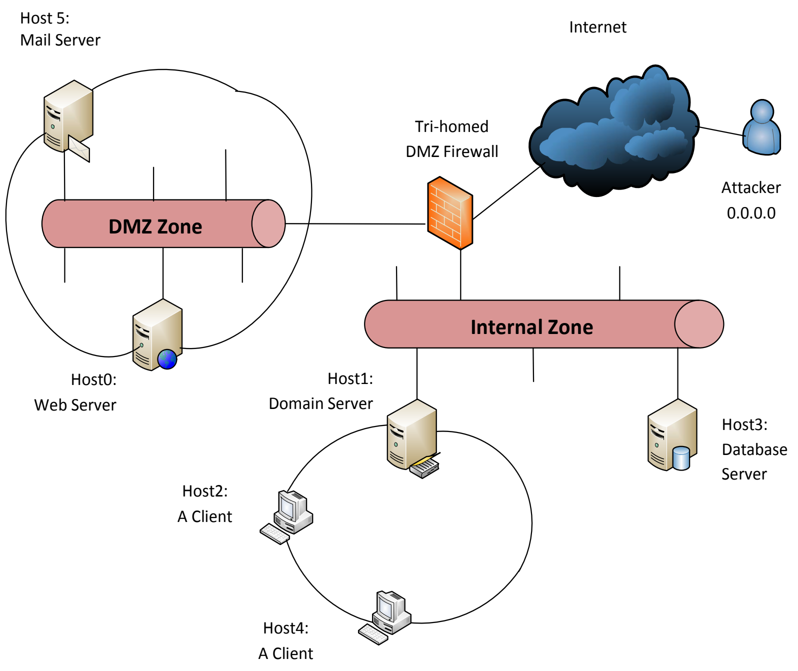


Figure 8.1: Test Network

$Host_3$ is the adversary's target machine and *MySQL* is the critical resource running over it. The adversary is an malicious entity and her goal is to obtain root level privileges on $Host_3$. Table 8.2 shows the system characteristics for the hosts available in the network. Such kind of information is available in public vulnerability

$Host$	$adversary$	$Host_0$	$Host_1$	$Host_2$	$Host_3$	$Host_4$	$Host_5$
$adversary$	Local-host	ALL	NONE	NONE	NONE	NONE	SMTP
$Host_0$	ALL	Local-host	ALL	Netbios-ssn OpenSSH	Squid LICQ	Squid LICQ	NONE
$Host_1$	ALL	IIS	Local-host	Netbios-ssn	Squid LICQ	NONE	NONE
$Host_2$	ALL	IIS	ALL	Local-host	Squid, LICQ MS SMV	ssh	SMTP
$Host_3$	ALL	IIS	ALL	ALL	Local-host	NONE	NONE
$Host_4$	ALL	NONE	NONE	NONE	NONE	Local-host	NONE
$Host_5$	ALL	NONE	NONE	NONE	NONE	NONE	Local-host

Table 8.1: Connectivity-Limiting Firewall Policies

Host	Services	Ports	Vulnerabilities	CVE IDs
$Host_0$	IIS web service	80	IIS buffer overflow	CVE-2010-2370
	ftp	21	ftp buffer overflow	CVE-2009-3023
$Host_1$	ftp	21	ftp rhost overwrite	CVE-2008-1396
	ssh	22	ssh buffer overflow	CVE-2002-1359
	rsh	514	rsh login	CVE-1999-0180
$Host_2$	netbios-ssn	139	netbios-ssn nullsession	CVE-2003-0661
	rsh	514	rsh login	CVE-1999-0180
	OpenSSH	22	Heap Corruption in OpenSSH	CVE-2003-0693
$Host_3$	LICQ	5190	LICQ-remote-to-user	CVE-2001-0439
	Squid proxy	80	squid-port-scan	CVE-2001-1030
	MySQL DB	3306	local-setuid-bof	CVE-2006-3368
	MS SMV Service	445	MS SMV Service Stack BoF	CVE-2008-4050
$Host_4$	LICQ	5190	LICQ-remote-to-user	CVE-2001-0439
	Squid proxy	80	squid-port-scan	CVE-2001-1030
	ssh	22	ssh buffer overflow	CVE-2002-1359
$Host_5$	SMTP	25, 143	SMTP Remote Code Execution	CVE-2004-0840
	Squid proxy	80	squid-port-scan	CVE-2001-1030

Table 8.2: System Characteristics for the Test Network

databases viz. [215], [216], etc. The attack graph, generated for the running example using MulVAL tool ([217]), is shown in the Figure 8.2, where an exploit is shown by an oval, an initial condition is shown by a box, and a postcondition is shown by a simple plain-text. There are many cycles in the attack graph, but for easier illustration, we consider two of them: (*node37*→*node29*→*node35*→*node32*→*node37*) and (*node19*→*node37*→*node29* →*node27*→*node25* →*node19*). These two cycles in the attack graph are depicted using bold arrows.

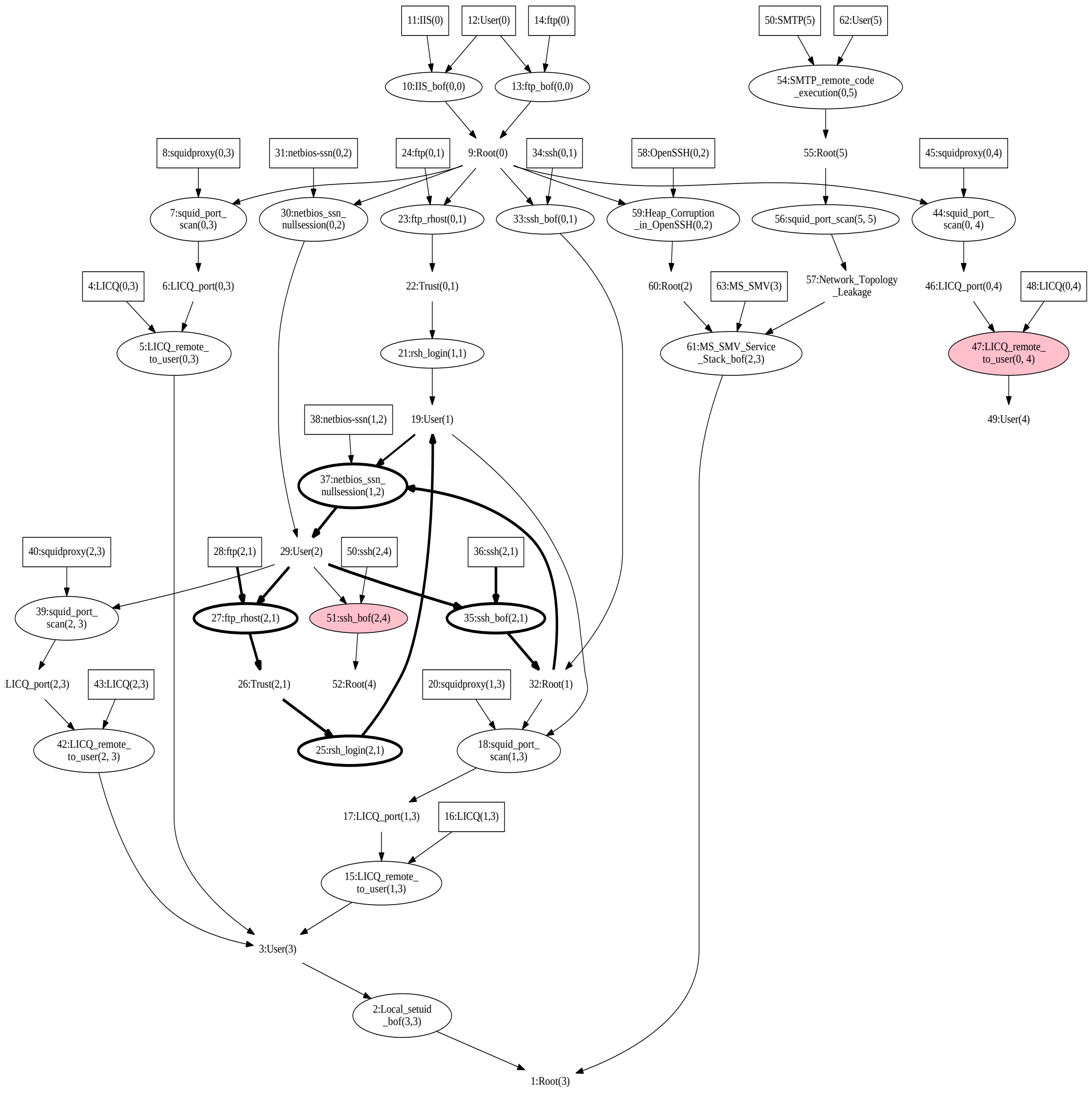


Figure 8.2: Goal-Oriented Attack Graph for the Test Network

As per the monotonicity assumption ([218]), the adversary never relinquishes her privileges on the previously compromised hosts. Therefore, in a realistic scenario, the adversary does not follow loop attack paths as she does not strive for already acquired privileges. Being aware of this point, an administrator makes effort to find all non-loop attack paths to a target resource. Since the presence of a cycle in the attack graph complicates quantitative analysis, existing multistage attack modeling techniques ([217, 218, 219, 220]) apply cycle detection algorithms. However, detection of a cycle in a large attack graph is computationally expensive.

8.2 Recursive Composition Algebra for Multistage Attack Modeling

While applying RCA (described in Chapter 3) for the multistage attack modeling, we make the following changes: (i) exploits are the operands instead of Web services, and (ii) service-message tuple is replaced with *exploit-preconditions* tuple and operators are redefined based on it. Further, in order to build intuition about the modeling, we make the following assumptions:

1. An adversary is a skilled intruder, external to the network, whose goal is to gain illegitimate access to the enterprise resources. Moreover, we assume that she is able to successfully exploit all the vulnerabilities present in the network.

2. Since the adversary does not gain any sort of direct access into the network by performing a *Denial of service* (DoS) attack, we do not consider the vulnerabilities related to the attacks on availability.

8.2.1 Exploit

Let $\mathbb{H} = \{h_1, h_2, \cdots, h_m\}$ be a finite set of hosts in an enterprise network that can be potential targets for the adversary and let $\mathbb{V} = \{v_1, v_2, v_3, \cdots, v_n\}$ be a finite set of vulnerabilities present on the vulnerable hosts. Let $\mathbb{E} = \{e_1, e_2, \cdots, e_n, \epsilon\}$ be a finite set of exploits that can take advantage of vulnerabilities in the set $\mathbb{V}$. An empty exploit ϵ never exploits any of the vulnerability present in $\mathbb{V}$ and cannot be invokable from any other exploit. We define an exploit $e_i \in \mathbb{E}$ as follows:

Definition 8.2.1 (Exploit). An exploit $e_i \in \mathbb{E}$ is a 3-tuple $\langle I, O, Rl \rangle$, where $I = \{I_1, \cdots I_p\},\ p \in \mathbb{N}$ is a finite set of preconditions, that e_i requires in order to be executed successfully. $O = \{O_1, \cdots, O_q\},\ q \in \mathbb{N}$ is a finite set of postconditions, that e_i produces once executed successfully. Rl is a relation that maps preconditions from I to postconditions in O ($Rl \subset 2^I \times O$). $e_i.I$ and $e_i.O$ are referred as the set of preconditions and the set of postconditions for the exploit e_i.

Based on the relationship between security conditions (preconditions and postconditions) and exploits, the relation Rl can be split into *require relation* and *imply relation*. An exploit and its preconditions are related by *require relation* which states that for successful execution of an exploit, all of its preconditions need to be satisfied conjunctively. Given an exploit e_i, let $e_i.I = \{pre_1, pre_2, pre_3\}$ and $e_i.O = \{post_1, post_2, post_3\}$ be the sets of preconditions and postconditions respectively. Then $e_i.Rl$ is the relation that maps all the preconditions from $e_i.I$ to one of the postconditions in $e_i.O$, i.e. $e_i.Rl = \langle pre_1 \wedge pre_2 \wedge pre_3, post_1 \rangle$ or $\langle pre_1 \wedge pre_2 \wedge pre_3, post_2 \rangle$ or $\langle pre_1 \wedge pre_2 \wedge pre_3, post_3 \rangle$.

Example 8.2.1. In our running example (Fig. 8.2), $IIS(0)$ and $User(0)$ are the two preconditions for the exploit $IIS\_bof(0,0)$ (node no. 10) and $Root(0)$ is the implied postcondition.

$$\begin{aligned} IIS\_bof(0,0).I &= (IIS(0), User(0)) \\ IIS\_bof(0,0).O &= (Root(0)) \\ IIS\_bof(0,0).Rl &= \langle IIS(0) \wedge User(0), Root(0) \rangle \end{aligned}$$

An exploit can be engineered in many different ways to get the advantage of an exposed vulnerability. Therefore, based on the goal of the adversary, consequences of successful exploitation of a particular vulnerability could be many ranging from the accidental disclosure of non-relevant information to fully privileged remote access to a critical system. The consequences may be any of the following: increased connectivity, escalated privileges, and increased vulnerabilities. One of the ultimate goals of an expert adversary is to establish a foothold in order to maintain control of the compromised hosts even if the user logs off or the computer reboots. This type of maintaining *persistence control* can be achieved by installing rootkits/backdoors, creating new services, new scheduled tasks, modifying registry keys, so the malicious service starts at next boot. An *imply relation* exists between an exploit and its postconditions. Successful exploitation of a vulnerability leads to the generation of the said postconditions disjunctively. Further, the implied postconditions may act as preconditions for other exploits.

8.2.2 Operators

Definition 8.2.2 (Absolute successor)**.** Let '$\succ$' be a symbol to represent the successor operator. $\succ$ maps an element of the $\mathbb{E}$ to an element of the power set of the set $\mathbb{E}$ ($\succ$: $\mathbb{E} \rightarrow 2^{\mathbb{E}}$). Let $S \subset \mathbb{E}$ and $e_i \in \mathbb{E}$, then $\succ (e_i) = S$ if and only if $\forall e_j \in S \ : \ e_i.O \cap e_j.I \neq \emptyset$.

In other words, the absolute successor operator ($\succ$) is an unary operator that provides the exploit(s) directly invokable by a given exploit. Such invoked exploit(s) are called as successor exploit(s). The successor operator captures the dependency among exploits. The dependency between two exploits is satisfied if all the initial conditions vital for the successful exploitation of a dependent (invoked) exploit are satisfied by the dependency. Exploit dependencies are made explicit by invoking other exploit(s) from the given exploit. Such dependency is possible when a post-condition of an exploit is proved to be one of the necessary preconditions for other exploit(s) provided that the remaining preconditions are already satisfied.

Consider that an exploit $e_i \in \mathbb{E}$ invokes another exploit $e_j \in \mathbb{E}$. Then $\succ (e_i) = e_j$. If e_j is not known in advance, we write $\succ (e_i) = e_{i+1}$ unless stated otherwise. If e_i directly invokes a set of exploits $(e_1, \cdots, e_l) \subset \mathbb{E}$, then $\succ (e_i) = \{e_1, \cdots, e_l\}$. If e_i does not invoke any exploit from the set $\mathbb{E}$, then $\succ (e_i) = \epsilon$.

Nowadays, Cyber attacks combine multiple exploits in order to get incremental access to network resources. The composition of exploits (say $n \in \mathbb{N}$, number of exploits) is the combination of multiple host-only exploits into a meta exploit based on their require/imply relationship. The exploits can be composed either sequentially or parallely. Let '$\oplus_s$' and '$\oplus_p$' be two symbols that represent the sequential composition and parallel composition respectively. We define sequential and parallel composition as follows.

Definition 8.2.3 (Sequential composition)**.** Given two exploits $e_i, e_j \in \mathbb{E} : e_j \in (\succ e_i)$, sequential composition of e_i and e_j (represented as $e_i \oplus_s e_j$) yields a meta-exploit e_k such that the preconditions of e_k matches with the preconditions of e_i and the postcondition of e_k matches with the postcondition of e_j.

$$e_i \oplus_s e_j \triangleq \{e_k : (e_k.I = e_i.I) \wedge (e_k.O = e_j.O)\} \tag{8.1}$$

Figure 8.3 depicts a scenario (from the running example) that composes exploits $IIS\_bof(0,0)$ and $ssh\_bof(0,1)$ sequentially.

Definition 8.2.4 (Parallel composition)**.** Given two exploits $e_i, e_j \in \mathbb{E} : e_j \notin (\succ e_i) \wedge e_i \notin (\succ e_j)$, parallel composition of e_i and e_j (represented as $e_i \oplus_p e_j$) yields a composite exploit e_k such that the preconditions of e_k is consolidation of the preconditions of e_i and e_j and the postcondition of e_k is consolidation of the postconditions of e_i and e_j.

$$e_i \oplus_p e_j \triangleq \{e_k : (e_k.I = (e_i.I \cup e_j.I)) \wedge (e_k.O = (e_i.O \cup e_j.O))\} \tag{8.2}$$

Parallel composition of exploits in multistage attack points towards the possibility of coordinated attack. In order to improve the chances of successful attack, two or more attackers controlling different hosts (in a target network) may collude and cooperate towards achieving a common goal [221].

Example 8.2.2. If an exploit e_i implies a postcondition o_1 and an exploit e_j implies a postcondition o_2, and both o_1 and o_2 are required by an exploit e_s, then e_s cannot be executed before e_i and e_j are executed. More than one adversary could coordinate to execute e_i and e_j at a time.

In case of parallel composition, an adversary has to compulsorily execute all of the participating exploits. The postconditions generated by the participating exploits become the preconditions for the successor exploit. Figure 8.4 depicts a parallel composition scenario for the running example. Let '$\oplus$' be a symbol for common representation for both sequential composition operator and parallel composition operator (removing the suffixes s and p from $\oplus_s$ and $\oplus_p$). Let $e_i, e_j \in \mathbb{E}$ be two exploits such that their composition $(e_i \oplus e_j)$ is possible. However, the resultant exploit (e_k) for the composition does not exist in the set $\mathbb{E}$. Then, '$e_i \oplus e_j$' itself represents a composite exploit that is able to participate in further composition processes as a single exploit. However, the composition of an exploit with an empty exploit results in the exploit itself $(e_i \oplus \epsilon = e_i)$.

Definition 8.2.5 (Conditional successor)**.** A conditional successor operator (represented as '$\succ_C$') accepts the input and produces the output in the form of a tuple $\langle e_i, I_p \rangle$, where $e_i \in \mathbb{E}$ and I_p is a set of preconditions for e_i. Given a tuple $\langle e_i, I_p \rangle$, $\langle e_j, I_r \rangle$ is a conditional successor of $\langle e_i, I_p \rangle$ $\Big($written as: $\langle e_j, I_r \rangle \in \Big(\succ_C \langle e_i, I_p \rangle \Big)\Big)$ if and only if $e_j \in \Big(\succ e_i \Big)$ and $I_r \in e_i.Rl(I_p)$.

Here, $\succ_C$ operator accepts input in the form of pair $\langle e_i, I_p \rangle$, where e_i is the exploit and I_p is the set of preconditions required for successful exploitation of e_i. All

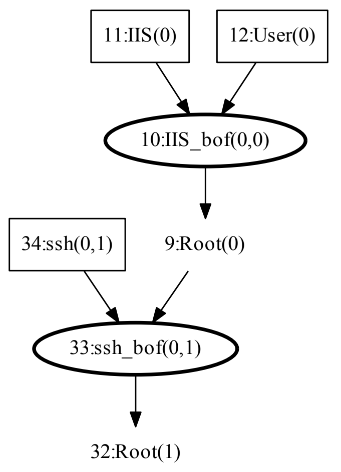


Figure 8.3: Sequential Composition of Exploits

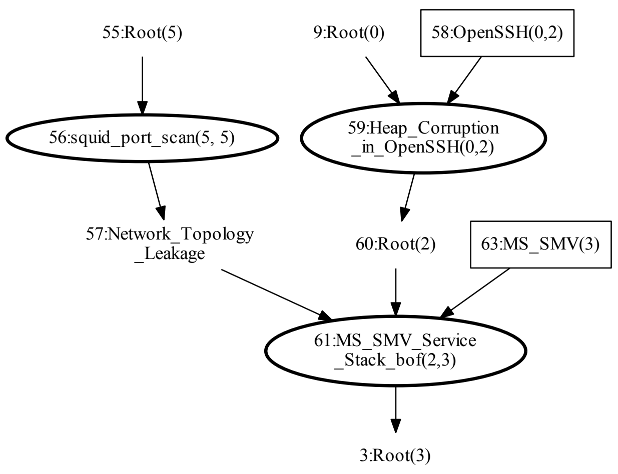

55:Root(5)
9:Root(0)
58:OpenSSH(0,2)
56:squid_port_scan(5, 5)
59:Heap_Corruption_in_OpenSSH(0,2)
57:Network_Topology_Leakage
60:Root(2)
63:MS_SMV(3)
61:MS_SMV_Service_Stack_bof(2,3)
3:Root(3)

Figure 8.4: Parallel Composition of Exploits

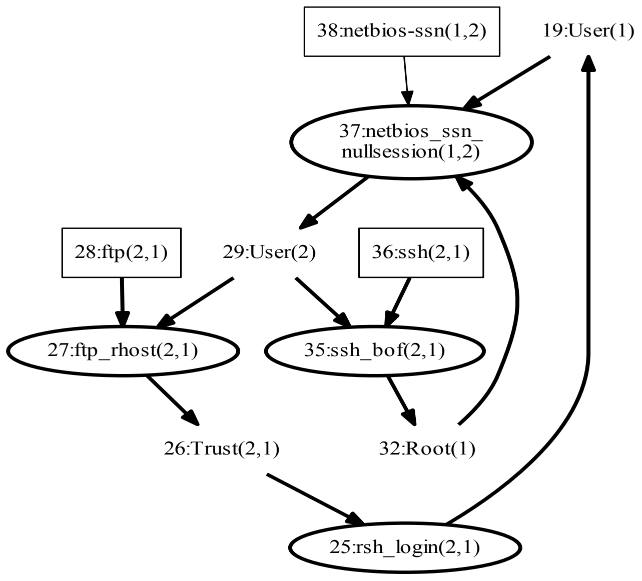


Figure 8.5: Cycles in the Attack Graph

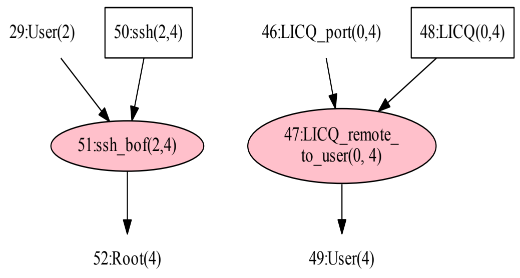


Figure 8.6: Dead Ends in the Attack Graph

the preconditions should be conjunctively satisfied for e_i to be executed successfully. The notion of $\succ_C$ arises from the AND relationship between preconditions of an exploit e_i. The conditional successor of e_i produces an output pair $\langle e_j, I_r \rangle$ where I_r is the (set of) postcondition(s) generated by e_i. There is a disjunctive OR relationship between the generated postconditions and they act as one of the preconditions required for the successful exploitation of e_j.

Restrictive successor operator ($\succ_R$) is a conditional successor operator such that $Domain(\succ_R) = Domain(\succ)$ and $Range(\succ_R) \subseteq Range(\succ_C)$. The rational behind using $\succ_R$ operator is to realize the notion of monotonicity ([218]) in the multistage attacks which states that an adversary's control over the network increases monotonically. In other words, the adversary need not relinquish her privileges on the already compromised resources while advancing further in the network. The notion of monotonicity allows all potential network attacks to be represented as a sequence of dependencies among exploits and security conditions, rather than as an enumeration of states. We formally define the restrictive successor operator as follows:

Definition 8.2.6 (Restrictive successor)**.** Let $\langle e_i, I_p\rangle \oplus \langle e_j, I_q\rangle \oplus \cdots \oplus \langle e_n, I_s\rangle$ be an exploit composition chain and $\langle e_x, I_x\rangle$ be an exploit-precondition tuple, then $\langle e_x, I_x\rangle \in \Big(\succ_R \langle e_i, I_p\rangle \oplus \langle e_j, I_q\rangle \oplus \cdots \oplus \langle e_n, I_s\rangle\Big)$ if and only if $\langle e_x, I_x\rangle \in \Big(\succ_C \langle e_i, I_p\rangle \oplus \langle e_j, I_q\rangle \oplus \cdots \oplus \langle e_n, I_s\rangle\Big)$ and $\langle e_x, I_x\rangle \notin \{\langle e_i, I_p\rangle, \langle e_j, I_q\rangle, \cdots, \langle e_n, I_s\rangle\}$.

The advantage of $\succ_R$ is that it eliminates the cycles in the resulting graph. Algorithm 8.1 presents the process for computing a restrictive successor.

Algorithm 8.1 ResSuc($\langle e_i, I_p\rangle \oplus \langle e_j, I_q\rangle \oplus \cdots \oplus \langle e_n, I_s\rangle, \mathbb{E}$)

```
    Input: $\langle e_i, I_p\rangle \oplus \langle e_j, I_q\rangle \oplus \cdots \oplus \langle e_n, I_s\rangle, \mathbb{E}$
    Output: Restrictive successors of $\langle e_i, I_p\rangle \oplus \langle e_j, I_q\rangle \oplus \cdots \oplus \langle e_n, I_s\rangle$
1:  for all $e_j \in e_i \oplus \cdots \oplus e_n$ do
2:      $\mathbb{E} \leftarrow \mathbb{E} - \{e_j\}$
3:  end for
4:  for all $e_k \in \mathbb{E}$ do
5:      if $e_n.Rl(I_s) \in e_k.I$ then
6:          $\Big\langle e_k, \Big(e_n.Rl(I_s)\Big)\Big\rangle \in \Big(\succ_R (\langle e_i, I_p\rangle \oplus \langle e_j, I_q\rangle \oplus \cdots \oplus \langle e_n, I_s\rangle)\Big)$
7:      end if
8:  end for
```

Let '⊛' be a symbol to represent the recursive composition. To define recursive composition, we incorporate a restrictive successor operator ($\succ_R$) and a composition operator ($\oplus$) as supplementary operators (defined earlier in this section).

Definition 8.2.7 (Recursive composition)**.** Recursive composition for a given exploit-precondition tuple $\langle e_i, I_p\rangle$ is defined as follows:

$$\circledast \langle e_i, I_p\rangle \triangleq \begin{cases} \epsilon & ;\text{if } \langle e_i, I_p\rangle = \epsilon \\ \langle e_i, I_p\rangle & ;\text{if } \succ_R \langle e_i, I_p\rangle = \epsilon \\ \langle e_i, I_p\rangle \oplus \Big\{ \circledast (\succ_R \langle e_i, I_p\rangle)\Big\} & ;\text{otherwise} \end{cases} \tag{8.3}$$

Algorithm 8.2 presents the process for computing the recursive composition among services.

Algorithm 8.2 RecComp($\langle e_i, I_p\rangle, \mathbb{E}$)

```
    Input: $\langle e_i, I_p\rangle, \mathbb{E}$
    Output: RCG with $\langle e_i, I_p\rangle$ as root
 1: make $\langle e_i, I_p\rangle$ as root node
 2: $ParentNode \leftarrow \langle e_i, I_p\rangle$
 3: $\mathbb{E} \leftarrow \mathbb{E} - \{e_i\}$
 4: $S \leftarrow \emptyset$
 5: for all $e_j \in \mathbb{E}$ do
 6:     $\mathcal{R} \leftarrow ResSuc(\langle e_i, I_p\rangle, \mathbb{E})$
 7:     if $\langle e_j, I_x\rangle \in \mathcal{R}$ then
 8:         $e_i \oplus e_j$
 9:         $ParentNode.Child \leftarrow \langle e_j, I_x\rangle$
10:         $S \leftarrow (\langle e_i, I_p\rangle \oplus \langle e_j, I_x\rangle)$
11:         $\mathbb{E} \leftarrow \mathbb{E} - \{e_j\}$
12:     end if
13: end for
14: while $S \neq \emptyset$ do
15:     for all $\langle e_i, I_p\rangle \in S$ do
16:         RecComp($\langle e_i, I_p\rangle, \mathbb{E}$)
17:     end for
18: end while
```

Recursive composition on e_i generates a topologically sorted directed acyclic graph with e_i as the root node. We call every path (from the root to the sink node) in the graph as a *attack trace*. Let e_i be an exploit then $\mathcal{T}_{e_i}$ represents a set which contains all the attack traces generated by applying the recursive composition on e_i. Here, we are not discussing algebraic properties of the operators as they have been already discussed in Chapter 3.

8.3 Case Study

We apply RCA over a test network shown in Figure 8.1. We assume that the adversary is a skilled malicious entity on the Internet, capable of successfully exploiting all the vulnerabilities present in the network, and her goal is to obtain root privilege on the database server (i.e. Host 3). Initially, the adversary has an access to the services running over the Web Server hosted in the DMZ. The XML specifications for the threat agent (adversary), host(s) and network configuration (service connectivity), and exploit description are given in Figure 8.7, 8.8, and 8.9 respectively. As shown in Figure 8.8, host(s) and network description capture information about the network hosts, services running over them, service connectivity between hosts, and vulnerabilities present in the network. Service connectivity information is obtained from the firewall rules (access control policies). Vulnerability scanners such as [222], [223], [224] etc., can be used to obtain information about the vulnerabilities present in the network.

Figure 8.7 illustrates the XML specification for the threat agent. Adversary capabilities uses two XML tags: *position* and *entry_point*. Position states about the adversary's initial location from where she start exploiting entry point vulnerabilities present in the Internet facing applications.

```
<!--Attacker Capabilities-->

<position>attacker.internet</position>
<entry_point no="1">attacker.Web_Server</entry_point>
<entry_point no="2">attacker.Mail_Server</entry_point>
```

Figure 8.7: Adversary Capabilities

```xml
<!-- Host Description-->
<host name=“Web_Server" network="dmz">
 <services>
         <IIS_Web_Service/>
         <ftp/>
 </services>
 <connectivity>
         <remote id="1"> <IIS/> </remote>
         <remote id="2"> <ftp/> </remote>
 </connectivity>
 <cve>
         <CVE_2010_2370/>
         <CVE_2009_3023/>
 </cve>
</host>
```

Figure 8.8: Network Description

Figure 8.9 shows exploit description for all the vulnerabilities present in the network. Such kind of information is available in the well-known vulnerability databases such as [215], [216], etc. Such vulnerability databases provides several distinctions between vulnerabilities such as the necessary position of an attacker relative to the target host and the consequence of an exploited vulnerability. As shown in Figure 8.9, exploit description uses two XML tags: *preconditions* and *postcondition*. Prior to the application of RCA, an administrator needs to obtain system information as well as vulnerability information and evaluate each of the discovered vulnerabilities for their preconditions and postconditions.

The exploit composition phase leverages the XML specifications (depicted in the Figure 8.7, 8.8, 8.9) via the recursive composition algorithm (Algorithm 8.2) discussed in Section 8.2. Recursive composition yields exhaustive recursive composition graph (RCG) as shown in Fig. 8.10. The RCG enumerates all potential multistage attack scenarios for the test network and is free from the cycles. Each attack trace is a series of exploits that leads to an undesirable state (i.e. the state where an adversary can obtain user/root level privileges). Once we have a RCG for a specific network, we can utilize it for further analysis. Similar to a attack graph, the generated RCG can be used for (i) attack forecasting ([225]) (ii) cost-benefit security hardening ([226]) (iii) evaluating the impact of network infrastructure factors such as network segregation/partitioning, defense-in- depth, service connectivity, etc., on the network security risk ([227]), and (iv) measuring the temporal variation in the

```xml
<!-- Exploit Description-->
<exploit name=“IIS_bof" cve=“CVE_2010_2370">
 <preconditions>
          <privilege host=“Source" rel=“gte” value=“user”/>
          <privilege host=“Web_Server” rel=“eq” value=“user”/>
   <service_connectivity from=“source” to=“Web_Server” service=“IIS_Web_Service”/>
 </preconditions>
 <postcondition>
          <privilege host=“Web_Server” value=“root”/>
 </postcondition>
</exploit>

<exploit name=“ftp_bof" cve=“CVE_2009_3023">
 <preconditions>
          <privilege host=“Source" rel=“gte” value=“user”/>
          <privilege host=“Web_Server” rel=“eq” value=“user”/>
   <service_connectivity from=“source” to=“Web_Server” service=“ftp”/>
 </preconditions>
 <postcondition>
          <privilege host=“Web_Server” value=“root”/>
 </postcondition>
</exploit>
```

Figure 8.9: Exploit (Vulnerability) Description

network security risk ([228], [229]). The logical inferences derived from the RCA are discussed in the next subsection. The derived inferences help administrators to take proactive actions against the possible attacks.

8.3.1 Logical Inferences

Once we have the RCG for a network under consideration, we can perform further analysis to extract security relevant information for proactive network hardening. For doing this, we use the concept of canonical sets of exploits derived from the RCG. The term *canonical* is not an absolute one. It gives meaning to the word adjoining it. We define the term *canonical set* for an exploit (in the context of RCA) as follows.

Definition 8.3.1 (Canonical set for an exploit e_i)**.** Given the set $\mathbb{E}$, a canonical set $\mathcal{C}_i$ for an exploit $e_i \in \mathbb{E}$ is a subset of $\mathbb{E}$ such that it consists of all sink nodes (other than

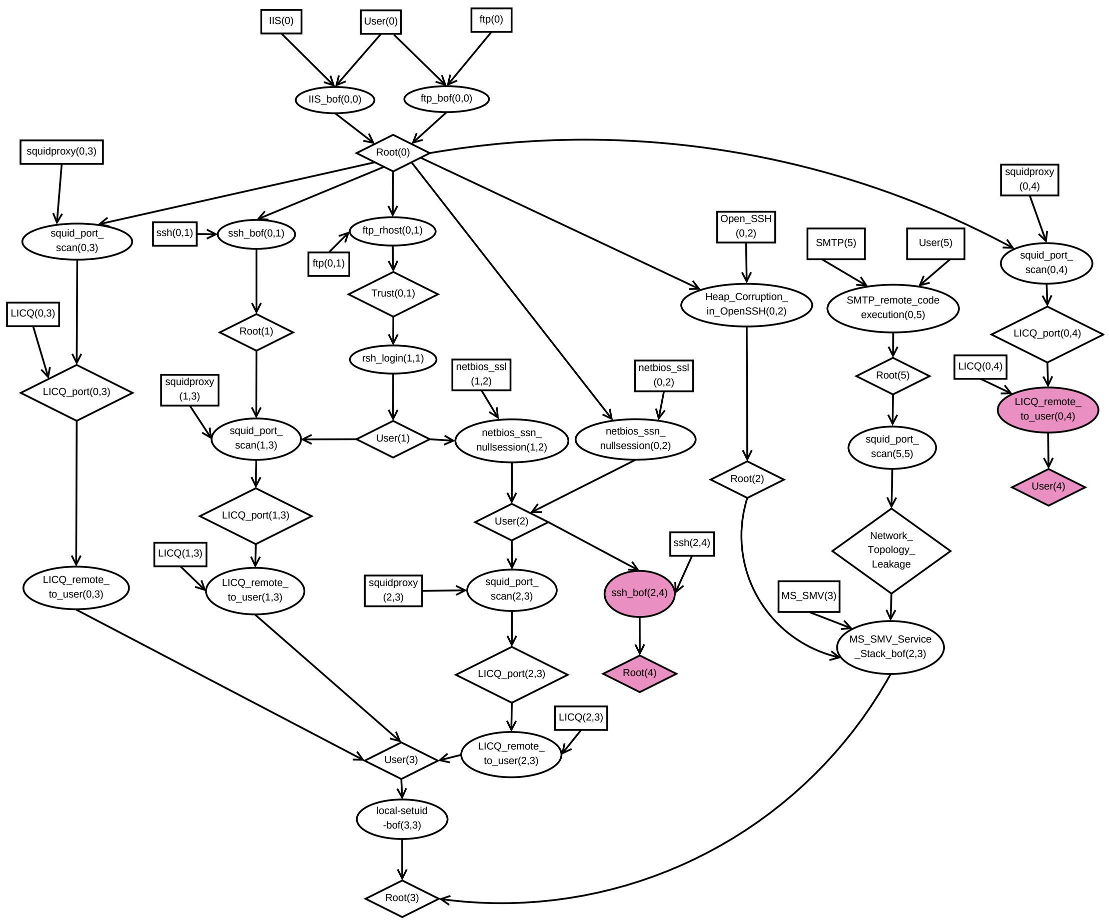


Figure 8.10: RCG for the Test Network

the root node) of a RCG generated from the application of the recursive composition operation ($\circledast$) on the exploit e_i.

Let $\mathcal{C}_i$ be a canonical set for an exploit e_i and '$\rightsquigarrow$' be a symbol to represent '*leads to*'. Then, $\circledast e_i \rightsquigarrow \mathcal{C}_i$. Even if an exploit does not invoke any other exploit, an empty canonical set exists for it. For instance, if $\circledast\, e_i = \epsilon$ then $\circledast\, e_i \rightsquigarrow \mathcal{C}_i = \emptyset$. The computation of canonical sets for all the exploits yields a set $\mathbb{C}$ of the set $\mathbb{E}$. The partition set $\mathbb{C}$ (may be non-disjoint) consists 'n' number of sets $\mathbb{C} = (\mathcal{C}_1, ..., \mathcal{C}_n)$. $\mathcal{C}_i$ is the canonical set generated by e_i where $0 < i \leq n$.
Let S_{is}, S_{ig}, and S_{tm} be the sets of isolated, strict igniter, and strict terminator exploits, respectively. Several logical interpretations based on the canonical sets are deduced and discussed with their significance as follows:

8.3.1.1 Isolated Exploit

A non-trivial isolated exploit is one that cannot be invoked by any other exploit as well as it cannot invoke other exploits. Excluding isolated exploits (vulnerabilities) out from $\mathbb{E}$ is mandatory as their presence in the exploit set $\mathbb{E}$ increases the computational overhead during exploit composition. On the basis of recursive composition operator and canonical sets, isolated exploits can be recognized automatically as follows:

$$(\exists e_i \in \mathbb{E}) \left[(\nexists e_j \in \mathbb{E}) \left((\circledast e_i \rightsquigarrow \mathcal{C}_i = \emptyset) \wedge (\circledast e_j \rightsquigarrow \mathcal{C}_j) \wedge (e_i \in \mathcal{C}_j) \right) \right] \Leftrightarrow e_i \in S_{is} \tag{8.4}$$

In practice, for a computer network of reasonable size, vulnerability scanners generate an overwhelming amount of data in the form of laundry list of vulnerabilities. Patching all reported vulnerabilities in a network is mission impossible for the administrator as it costs money, time, resources, etc., so where does an administrator start? If there are so many vulnerabilities to fix, the administrator needs to identify the vulnerabilities that really matter most in securing the network. Due to the absence of one or more enabling conditions, some of the vulnerabilities in a network are not exploitable. Therefore, in today's resource-constrained network environment, patching of such temporarily inactive vulnerabilities is of no value. One needs to focus on a group of exploitable vulnerabilities that endangers the network security. Identification of isolated vulnerabilities (temporarily inactive) reduces administrator search space and thereby help in cost-effective network hardening. For running example (shown in Figure 8.1), the vulnerability *CVE-1999-0180* in *rsh* service running over the $Host_2$ is the isolated vulnerability. Even though the *rsh* service is vulnerable, the vulnerability is not exploitable. It is because the service is not accessible to any other hosts.

8.3.1.2 Strict Igniter Exploit

A strict igniter exploit is one that cannot be invoked by another exploit but can invoke other exploits.

$$(\exists e_i \in \mathbb{E}) \left[(\nexists e_j \in \mathbb{E}) \left((\circledast e_i \rightsquigarrow \mathcal{C}_i \neq \emptyset) \wedge (\circledast e_j \rightsquigarrow \mathcal{C}_j) \wedge (e_i \in \mathcal{C}_j) \right) \right] \Leftrightarrow e_i \in S_{ig} \tag{8.5}$$

Algorithm 8.3 COMPUTING-LOGICAL-INFERENCES($\mathbb{E}, \succ, \oplus, \circledast$)

```
    **Input:** $\mathbb{E} = \{e_1, \cdots, e_n, \epsilon\}, \succ, \oplus, \circledast$
    **Output:** Logical inferences
 1: let $\mathbb{C} = \emptyset$ be a set
 2: for all $e_i \in \mathbb{E}$ do
 3:     $\circledast e_i \rightsquigarrow \mathcal{C}_i$        ▷ Derive canonical set for each exploit in the network.
 4:     $\mathbb{C} \leftarrow \mathcal{C}_i$
 5: end for
 6: let $S_{tm} = \emptyset$, $S_{is} = \emptyset$, $S_{ig} = \emptyset$ be the sets of strict terminator exploits, isolated
    exploits and strict igniter exploits, respectively.
 7: for all $\mathcal{C}_i \in \mathbb{C}$ do
 8:     if $\mathcal{C}_i = \emptyset$ then
 9:         for all $\mathcal{C}_j \in \mathbb{C}$ do
10:             if $e_i \in \mathcal{C}_j$ then
11:                 $S_{tm} \leftarrow e_i$        ▷ Strict terminator exploit
12:             else
13:                 $S_{is} \leftarrow e_i$    ▷ Isolated exploit (non-exploitable vulnerability)
14:             end if
15:         end for
16:     else if $\nexists \mathcal{C}_j \in \mathbb{C}$ *such that* $e_i \in \mathcal{C}_j$ then
17:         $S_{ig} \leftarrow e_i$        ▷ Strict igniter exploit (entry-point vulnerability)
18:     end if
19: end for
20: $\mathbb{E} \leftarrow \mathbb{E} \backslash S_{is}$        ▷ Exclude non-exploitable vulnerabilities
21: for all $e_i \in S_{is}$ do
22:     $\circledast e_i \rightsquigarrow \mathcal{C}_i$    ▷ Derive canonical set for non-exploitable vulnerabilities
23:     $\mathbb{C} \leftarrow \mathbb{C} \backslash \mathcal{C}_i$
24: end for
```

An adversary performs reconnaissance of the Internet facing hosts/servers in a target network by using tools like [222] or [230] and collects information about the vulnerabilities. One can call such vulnerabilities as entry point vulnerabilities. The adversary can enter into the network by exploiting entry point vulnerabilities where the exploited vulnerability lays the groundwork for the exploitation of subsequent

```
25: for all $\mathcal{C}_i \in \mathbb{C}$ do
26:     for all $\mathcal{C}_j \in \mathbb{C}$ do
27:         if $\mathcal{C}_i = \mathcal{C}_j$ then
28:             $\mathcal{T}_{e_i}$ and $\mathcal{T}_{e_j}$ behave alike (i.e. attack traces from $e_i$ and $e_j$ end up in the same set of resources)
29:         else if $\mathcal{C}_i \subset \mathcal{C}_j$ then
30:             if $e_i$ is a successor of $e_j$ then
31:                 preventing $e_j$ from execution also prevents $e_i$
32:             else
33:                 both $e_i$ and $e_j$ needs to be prevented from execution
34:             end if
35:         end if
36:     end for
37: end for
```

vulnerabilities. Identification of such entry point vulnerabilities (or strict igniter exploits) helps the administrator in closing all required doors leading to mission critical resources/assets.

8.3.1.3 Strict Terminator Exploit

A strict terminator exploit is one which does not invoke any exploit but can be invoked by other exploits.

$$(\exists e_i \in \mathbb{E}) \left[(\exists e_j \in \mathbb{E}) \left((\circledast e_i \rightsquigarrow \mathcal{C}_i = \emptyset) \land (\circledast e_j \rightsquigarrow \mathcal{C}_j) \land (e_i \in \mathcal{C}_j) \right) \right] \Leftrightarrow e_i \in S_{tm} \tag{8.6}$$

Classifying exploits into separate sets, namely strict igniter and strict terminator exploits reduces the search space while analyzing the exploit composition. By leveraging the condition $(\nexists e_j \in \mathbb{E})[e_i \in \mathcal{C}_j]$ and $(\exists e_j \in \mathbb{E})[e_i \in \mathcal{C}_j]$, strict igniter and strict terminator exploits can be computed.

Each attack trace starts from the adversary's initial position (i.e. the Internet) and ends up with either the target (i.e. critical resource of highest importance) or with the intermediate resource of less importance. Similar to [231], we define a dead end to be an exploit postcondition (adversary state) from which there is no attack path to the goal that the adversary wants to compromise. If the adversary follows the path that ends up with dead ends, then she will not be able to reach the desired target resource. Dead ends arise in the network because some host configurations cannot be penetrated, leaving no opportunity to the adversary to reach her goal.

Nowadays, it may not be possible for an adversary to reach the goal in all cases, because the adversary may get hold of the intermediate machines from which the target machine simply cannot be reached. Further, a dead end may arise if no exploit is applicable for the target machine. Therefore, if an adversary encounters a dead end along a attack path, she has to backtrack. As per the monotonicity assumption ([218]), the adversary never relinquishes obtained privileges on the compromised machines. After backtracking, the adversary needs to scan other reachable hosts for vulnerabilities. Once found, she may exploit the newly discovered vulnerabilities. In the process, the adversary spends significant amount of effort and time in exploiting dead end vulnerabilities. Maintaining more number of dead end vulnerabilities distract adversary from reaching the target. Consequently, the ongoing attack will be slowed down and an administrator will get enough time to take preventive measures. [232] proposed an attack surface expansion (ASE) mechanism that focuses on increasing the number of vulnerabilities (entry-point vulnerabilities, in particular) visible to an adversary so that she cannot easily identify the real internal attack surface. Similar to [232], an administrator can increase the number of dead end vulnerabilities from where the adversary cannot reach the desired target resource. Since the administrator has a direct access to all of the network components, she can discover all the dead end vulnerabilities through internal scanning. Therefore, instead of patching the dead end vulnerabilities, it is desirable to increase them. An administrator can selectively patch the most critical vulnerabilities in the network in a cost effective manner. Figure 8.6 depicts the dead ends in the RCG (Fig. 8.10) generated for the running example.

8.3.1.4 Relations

As an added advantage of the canonical sets, an administrator can compare two exploits in terms of their attack propagation. The canonical set for an exploit e_i indicates how deeper (sphere of attack propagation) the adversary can penetrate the network, considering e_i as the first exploit. In the resource-constrained environment where patching of all the vulnerabilities is not possible, an administrator prioritize the exploitable vulnerabilities and patches the most pressing ones. For finding most pressing vulnerabilities, she has to derive a relation between exploitable vulnerabilities. A relation represents a logical association between two or more vulnerabilities. The relation between vulnerabilities is of two types: (i) overlapping relation and (ii) mutually exclusive relation.

(I) Overlap. Two exploits (exploitable vulnerabilities) overlap if there exists common exploit(s) between their canonical sets. Two exploits e_i and e_j are said to be overlapping if Equation 8.7 is satisfied.

$$(\exists e_i \in \mathbb{E}) \left[(\exists e_j \in \mathbb{E}) \left((\circledast e_i \rightsquigarrow \mathcal{C}_i) \land (\circledast e_j \rightsquigarrow \mathcal{C}_j) \land (\mathcal{C}_i \cap \mathcal{C}_j \neq \emptyset) \right) \right] \tag{8.7}$$

Two possible cases of the overlapping relation are discussed as follows:

- **Case I**. e_i and e_j are two overlapping exploits such that e_i is a successor (need not be immediate successor) of e_j. In this case, preventing e_j from execution (i.e. patching of respective vulnerability) automatically stops e_i, whereas converse is not true in general.

- **Case II**. e_i and e_j are two overlapping exploits such that no successor-predecessor relationship exists between e_i and e_j. In this case, preventing e_i from execution does not affect the adversary's capability in reaching the target resources. Therefore, an administrator must prevent (stop) both the exploits from execution.

(II) Mutually Exclusive. Two exploits e_i and e_j are said to be mutually exclusive if $C_i \cap C_j = \emptyset$ in Equation 8.7. The sphere of influence of one exploit is independent of the other and vice versa. Therefore, an administrator must focus on both the exploits while designing the security remediation plan. Existence of the mutually exclusive relation between exploitable vulnerabilities assist the administrator in understanding the number of independent vulnerabilities present in the network.

Identification of the said relationships between exploits helps the administrator in reducing the vulnerability search space and hence results in efficient hardening of the network. Algorithm 8.3 presents the process for deriving the logical inferences from the generated RCG using the notion of canonical set.

8.3.1.5 Cycle Detection in Attack Graph

If an administrator wants to know about the cycles, if any, present in the network, she can obtain such knowledge based on the notion of canonical set. Exploits in an attack cycle states that one exploit is reachable from the other and vice versa. The fulfillment of the following condition infers that the attack traces generated by e_i ($\mathcal{T}_{e_i}$) and e_j ($\mathcal{T}_{e_j}$) lead to a cycle in the network.

$$(\exists e_i \in \mathbb{E}) \left[(\exists e_j \in \mathbb{E}) \left(((\circledast e_i \rightsquigarrow \mathcal{C}_i) \wedge (e_j \in \mathcal{C}_i)) \wedge ((\circledast e_j \rightsquigarrow \mathcal{C}_j) \wedge (e_i \in \mathcal{C}_j)\right)\right] \tag{8.8}$$

Figure 8.5 depicts the scenarios where cycles exist in the network.

8.4 Related Work

In order to get knowledge of all plausible multistage attacks in an enterprise network, the formal models based on the vulnerability composition have been widely studied. In particular, the formal models allow inexpensive security analysis without real experiments ([233]) and offer elegant solutions for network vulnerability assessment and hardening. In the context of multistage attack, fault trees ([234]), privilege graph ([235, 236]), attack graphs ([237, 238, 239, 240]), and vulnerability graphs ([218, 220]) are the most appropriate modeling methodologies. Barbara et al. [241] presented a complete overview of such modeling techniques. Most of these models are the natural extension of the *Threat Logic Tree* ([242]) in one or several dimensions. Each model has different features, goals, advantages, and disadvantages.

In reliability engineering, fault trees ([234]) are primarily proposed for system failure analysis. Even though fault trees are well suitable for modeling conjunction (AND logic) and disjunction (OR logic) of faults, they are not expressive enough to capture all possible system failure scenarios. On the other hand, attack trees ([243, 244, 245, 246, 247, 248, 249]) were proposed to find out likely attack scenarios which can result in the violation of the network security policy. As attack trees are the scenario based approaches, it is impossible to cover all likely attack scenarios with few numbers of attack trees if there are several potential targets in a given network. Further, an attack tree contains more subjective nodes and the amount of required information may not be available in real world. Therefore, the attack tree is expert-specific and applicable only to completely known scenarios. In addition to the said limitations, the attack tree does not capture an attack scenario where one node is having multiple parents (that is, one initial condition can invoke multiple exploits).

The privilege graph ([235, 236]) depicts the adversaries privilege escalation in a target network. In a privilege graph, a node represents a set of privileges on a set of network resources and an edge represents the privilege escalation through successful execution of one or more exploits. Attack graphs ([237, 238, 239, 250, 251]) are obtained from privilege graphs by exploring various ways that an adversary may obtain the required privileges. Essentially, the attack graphs capture the computational environment and ease the vulnerability analysis of a network. It establishes the cause-consequence relationship between the adversarial actions. Initially, the attack graphs were constructed manually by red teams ([252]). However, manual construction of an attack graph is tedious, error-prone, and impractical for moderate size networks. Phillips et al. [251] proposed an attack graph model, where a node represents state of the network and an edge represent the actions taken by an adversary during the attack. However, the approach in [251] suffers from similar kind of limitations as in the manual construction of attack graphs.

In order to alleviate the problem of manual construction of an attack graph, Sheyner et al. [238] used a model checking based technique. The usefulness of the model checking based attack graph generation technique depends on the granularity of the input specification. The finer the granularity is, better will be the coverage, but it introduces a large number of states and hence the well-known state explosion problem. The vulnerability graphs (exploit-dependency attack graphs) ([218, 220]) depict the cause-consequence relationship between exploitable network vulnerabilities and the privileges that are required for an adversary to incrementally compromise the target network. In contrast to the model checking based approach ([238]), the vulnerability graphs are more popular for their improved scalability and granularity ([217, 220]). In spite of all these proposals, the presence of a cycle in a attack graph complicates quantitative and qualitative analysis of that graph. To ease the analysis, an administrator has to apply cycle detection algorithms which are computation intensive. Similar to the attack graph, our proposition generates a recursive composition graph (RCG), however, the RCG is free from the cycles.

Templeton and Levitt [253] proposed the require/provide model for logical correlation of different kinds of atomic attacks. The require/provide model states that a multistage attack comprises of a sequence of attacks and the early stages of an attack prepares for the later stages. Templeton and Levitt [253] defined the system states using simple predicates and devised JIGSAW language for the attack correlation. However, the approach of developing predicates in [253] is not systematic. Pandey et al. [254] proposed an algebra for capability based attack correlation and discussed algebraic operations and relations between the capabilities. A new service dependency model for attack response evaluation is proposed by Kheir et al. [255] that enables the evaluation of intrusion and response impact. The ultimate goal of said approaches ([253, 254, 255]) is IDS alert correlation, whereas our approach deals with vulnerability composition. RCA uses the require/provide model proposed in [253, 254]. At the core of our exploit composition process, capabilities (privileges) obtained by an adversary from the successful exploitation of previous vulnerability are used to satisfy a prerequisite (one of the enabling preconditions) of subsequent vulnerabilities. Moreover, the expressiveness of RCA subsumes the expressiveness of require/provide model.

8.5 Summary

In this chapter, a multistage attack modeling approach based on RCA is presented. The RCA generates a recursive composition graph (RCG) which depicts all possible multistage attack scenarios. RCA supports the systematic analysis of multistage attacks. The logical inferences deduced from the canonical sets helps the administrator in extracting security relevant information for proactive network hardening. For optimal network immunization, one can consider all possible countermeasures for each vulnerability.

Chapter 9

Conclusions and Future Directions

Broadly, Web service modeling techniques can be divided into transition-based and algebra-based. A transition-based modeling technique perceives the whole system as states and transitions between them, whereas an algebra-based modeling technique perceives the whole system as operands and operators. Algebraic techniques provide rigorous and sound foundation for formal reasoning [179, 184], whereas transitional ones are good for visualization and automation [106, 126]. Our requirements for modeling and verification of Web services motivate us to use a technique that is fusion of the both techniques.

In this thesis, therefore, we propose and employ an algebra, namely Recursive Composition Algebra (RCA), for the Web services verification that captures the primitive characteristics of Web services, and captures the concept of states and transitions as well. For the verification of Web services composition, a RCA based Web services set partitioning technique is described in Chapter 4. This technique partitions a given set of candidate Web services into a number of subsets by using the restrictive successor operator, and thereby generating a Web services set partition (WSSP) graph. Further, by using WSSP and trace concepts, deadlock and behavioral equivalence between services are verified.

A framework for modeling and verification of Web service interactions is presented in Chapter 5 that consists of a RCA-based modeling technique, a requirement specification language, and a verification technique. By using RCA, the modeling technique generates a *recursive composition interaction graph (RCIG)* that also works as an interpretation model. The requirement specification language *recursive composition specification language (RCSL)* is used to capture the requirements of Web service interaction scenarios, and is completely interpretable on a RCIG model. The verification technique is based on the *possible trace phenomenon* and employs the RCIG as its interpretation model and the RCSL as its specification language.

In Chapter 6, our aim is twofold: (i) to present a deadlock-free graph-based service behavior modeling technique that captures the notion of recursive composition among services, and (ii) to verify the compositional equivalence between two WSCGs formed before and after a structural change caused by the change in the set of available services. In addition to these, we aim that the verification technique should not be based on the subgraph isomorphism or subgraph bisimulation concept. In order to achieve the mentioned goals, in Chapter 6, we employ RCA to specify a service and its behavior. The RCA's application on a service (or service-message tuple) generates a DAG, namely RCG, that works as a building block in the formation of a WSCG. Then, by using the WSCG, the concepts of *composition expression* and *canonical composition expression* are described. Based on these concepts, the compositional equivalence between WSCGs is verified. Instead of directly using the WSCGs in order to check compositional equivalence, and solve an NP-Complete subgraph matching problem, this approach reduces the compositional equivalence verification process to the subsumption checking process between two algebraic expressions.

We implemented all the three proposed verification techniques, and an integrated verification tool is developed as presented in Chapter 7. There are three modules in the tool: (i) composition verification module, provided with a set of Web services, verifies deadlock and behavioral equivalence between services, (ii) interaction verification module, provided with a service repository and a requirement specification written in RCSL, verifies whether the given specification holds or not, and (iii) compositional equivalence verification module, provided with two WSCGs, verifies whether they are compositionally equivalent or not.

RCA was originally proposed for the modeling and verification of Web services. However, in the domain of Cyber security, similar scenarios arise as an adversary combines multiple vulnerabilities to gain incremental access to enterprise critical resources. Therefore, based on RCA, we modeled and analyzed the multistage attacks as described in Chapter 8. The underlying idea behind using RCA for multistage attack modeling is that the composed vulnerabilities can participate in a further composition process as a single vulnerability. Adopting the concept of recursive composition in the modeling process reduces the computational complexity of determining all the possible attack paths, and facilitates the hierarchical aggregation of the vulnerabilities ([256]) at different levels.

9.1 Directions for Future Research

The following research directions are suggested for future:

- Although the interaction verification technique described in Chapter 5 is completely able to achieve its objectives, it still has two limitations: partially solved state explosion problem and non-consideration of *Quality of Service* properties. Therefore, addressing these limitations is one of our future works.

- The compositional equivalence verification technique described in Chapter 6 could be improved by providing the support for: (i) specifying requirements about compositional equivalence and verification of that, (ii) generic kind of communication model rather than being focused only on the Web services. The participating entities for this communication model could be software components or intelligent agents (as defined in the context of artificial intelligence [6]) and the model could be able to capture the multi-party asynchronous communication among the software components or intelligent agents.

References

[1] Khalid S Khan, Regina Kunz, Jos Kleijnen, and Gerd Antes. **Five steps to conducting a systematic review**. *Journal of the Royal Society of Medicine*, **96(3)**:118–121, 2003.

[2] Gopal N. Rai, G. R. Gangadharan, and Vineet Padmanabhan. **Algebraic Modeling and Verification of Web Service Composition**. In *Proc. of the 6th Int. Conf. on Ambient Systems, Networks and Technologies (ANT)*, pages 675–679, 2015.

[3] Gopal N. Rai and G. R. Gangadharan. **Set Partition and Trace Based Verification of Web Service Composition**. In *Proc. of the 6th Int. Conf. on Ambient Systems, Networks and Technologies (ANT)*, pages 278–285, 2015.

[4] Gopal N. Rai and G.R. Gangadharan. **Architectural Characterization of Web Service Interaction Verification**. In *Proc. of 3rd Int. Conf. on Advanced Computing, Networking and Informatics*, **44**, pages 447–456. Springer, 2016.

[5] Mike P Papazoglou. **Service-oriented computing: Concepts, characteristics and directions**. In *Proc. of the 4th Int. Conf. on Web Information Systems Engineering (WISE)*, pages 3–12. IEEE, 2003.

[6] Munindar P Singh and Michael N Huhns. *Service-oriented computing: semantics, processes, agents*. John Wiley & Sons, 2006.

[7] Daniel Austin, Abbie Barbir, Christopher Ferris, and Sharad Garg. **Web services architecture requirements**. Technical report, W3C, 2004.

[8] Schahram Dustdar and Wolfgang Schreiner. **A survey on web services composition**. *Int. Journal of web and grid services*, 1(1):1–30, 2005.

[9] Maximilian Röglinger. **Verification of Web Service Compositions: An Operationalization of Correctness and a Requirements Framework for Service-oriented Modeling Techniques**. *Business & Inform. Systems Engineering*, 1(6):429–437, 2009.

[10] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT press, 1999.

[11] Ehtesham Zahoor, Kashif Munir, Olivier Perrin, and Claude Godart. **A bounded model checking approach for the verification of web services composition**. *Int. Journal of Web Services Research (IJWSR)*, 10(4):62–81, 2013.

[12] Hongbing Wang, Chen Wang, and Yan Liu. **A Logic-Based Approach to Web Services Composition and Verification**. In *Proc. of the World Conf. on Services-II, SERVICES-2'09*, pages 103–110. IEEE, 2009.

[13] Nobul Reddy Goli and Vinod Pathari. **A general framework for automatic verification of web services**. In *Proc. of the Int. Conf. on Advanced Computing and Communications*, pages 445–448. IEEE, 2006.

[14] Jamal Bentahar, Hamdi Yahyaoui, Melissa Kova, and Zakaria Maamar. **Symbolic model checking composite Web services using operational and control behaviors**. *Expert Systems with Applications*, 40(2):508–522, 2013.

[15] Warda El Kholy, Jamal Bentahar, Mohamed El Menshawy, Hongyang Qu, and Rachida Dssouli. **Modeling and verifying choreographed multi-agent-based web service compositions regulated by commitment protocols**. *Expert Systems with Applications*, 41(16):7478–7494, 2014.

[16] Yongxin Zhao, Hao Xiao, Zheng Wang, Geguang Pu, and Ting Su. **The semantics and verification of timed service choreography**. *Int. Journal of Computer Mathematics*, 91(3):384–402, 2014.

[17] M Emilia Cambronero, Gregorio Díaz, Valentín Valero, and Enrique Martínez. **Validation and verification of web services choreographies by using timed automata**. *The Journal of Logic and Algebraic Programming*, 80(1):25–49, 2011.

[18] Youssou Kasse and Lynda Mokdad. **Quantitative verification for response times in composite Web service model**. In *Proc. of the IEEE Symp. on Computers and Communications (ISCC)*, pages 97–102. IEEE, 2011.

[19] Sirine Rebai, Hatem Hadj Kacem, Mohamed Karâa, Saul E Pomares, and Ahmed Hadj Kacem. **A service-oriented architecture (SOA) framework for choreography verification**. In *Proc. of the 14th Int. Conf. on Computer and Inform. Sci. (ICIS)*, pages 642–646. IEEE, 2015.

[20] Ilyass El Kassmi and Zahi Jarir. **Security Requirements in Web Service Composition: Formalization, Integration, and Verification**. In *Proc. of the 25th Int. Conf. on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 179–184. IEEE, 2016.

[21] James Lyle Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, 1981.

[22] Bartek Kiepuszewski, Arthur HM ter Hofstede, and Wil MP van der Aalst. **Fundamentals of control flow in workflows**. *Acta Informatica*, **39**(3):143–209, 2003.

[23] Chun Ouyang, Eric Verbeek, Wil MP Van Der Aalst, Stephan Breutel, Marlon Dumas, and Arthur HM Ter Hofstede. **Formal semantics and analysis of control flow in WS-BPEL**. *Science of computer programming*, **67**(2-3):162–198, 2007.

[24] Xitong Li, Yushun Fan, Quan Z Sheng, Zakaria Maamar, and Hongwei Zhu. **A Petri net approach to analyzing behavioral compatibility and similarity of web services**. *IEEE Trans. on Systems, Man and Cybernetics, Part A: Systems and Humans*, **41**(3):510–521, 2011.

[25] Kais Klai and Hanen Ochi. **Checking Compatibility of Web Services Behaviorally**. In *Proc. of the 5th Int. Conf. on Fundamentals of Software Engineering (FSEN)*, pages 267–282, 2013.

[26] Guisheng Fan, Huiqun Yu, Liqiong Chen, and Dongmei Liu. **Petri net based techniques for constructing reliable service composition**. *Journal of Systems and Software*, **86**(4):1089–1106, 2013.

[27] Y. Du, W. Tan, and M. Zhou. **Timed Compatibility Analysis of Web Service Composition: A Modular Approach Based on Petri Nets**. *IEEE Transactions on Automation Science and Engineering*, **11**(2):594–606, 2014.

[28] Joachim Peer. *Description and automated processing of web services*. PhD thesis, University of St. Gallen, 2006.

[29] Shiyong Lu, Arthur Bernstein, and Philip Lewis. **Automatic workflow verification and generation**. *Theoretical Computer Science*, **353**(1):71–92, 2006.

[30] Piergiorgio Bertoli, Marco Pistore, and Paolo Traverso. **Automated composition of web services via planning in asynchronous domains**. *Artificial Intelligence*, **174**(3):316–361, 2010.

[31] Ourania Hatzi, Dimitris Vrakas, Mara Nikolaidou, Nick Bassiliades, Dimosthenis Anagnostopoulos, and Ioannis Vlahavas. **An integrated approach to automated semantic web service composition through planning.** *IEEE Trans. on Services Computing*, **5**(3):319–332, 2012.

[32] Dan Wu, Bijan Parsia, Evren Sirin, James Hendler, and Dana Nau. **Automating DAML-S web services composition using SHOP2**. In *Proc. of the Int. Semantic Web Conf.*, pages 195–210. Springer, 2003.

[33] Sheila A. McIlraith and Tran Cao Son. **Adapting Golog for Composition of Semantic Web Services**. In *Proc. of the 8th Int. Conf. on Principles and Knowledge Representation and Reasoning (KR)*, pages 482–496, 2002.

[34] Evren Sirin, Bijan Parsia, Dan Wu, James Hendler, and Dana Nau. **HTN planning for Web Service composition using SHOP2**. *Web Semantics: Science, Services and Agents on the World Wide Web*, **1**(4):377–396, 2004.

[35] Robin Milner. *Communicating and mobile systems: the pi calculus*. Cambridge university press, 1999.

[36] Zhang Chi. **Research on web service interface extending and composition compatibility checking based on pi calculus**. In *Proc. of the 2008 Int. Symp. on Inform. Sci. and Engineering*, pages 533–537. IEEE, 2008.

[37] G. Salaun, L. Bordeaux, and M. Schaerf. **Describing and reasoning on Web services using process algebra**. In *Proc. of the IEEE Int. Conf. on Web Services (ICWS)*, pages 43–50, 2004.

[38] Gwen Salaün. **Analysis and Verification of Service Interaction Protocols - A Brief Survey**. In *Proc. of the Fourth Int. Workshop on Testing, Analysis and Verification of Web Software, TAV-WEB*, pages 75–86, 2010.

[39] BS Yun, JW Yan, and M Liu. **A formal verification for web service composition based on CCS**. In *Proc. of the Key Eng. Materials*, **392**, pages 330–334. Trans Tech Publ, 2009.

[40] Wing Lok Yeung, Ji Wang, and Wei Dong. **Verifying choreographic descriptions of web services based on csp**. In *Proc. of the Services Computing Workshops SCW'06*, pages 97–104. IEEE, 2006.

[41] Rania Khalaf, Nirmal Mukhi, and Sanjiva Weerawarana. **Service-Oriented Composition in BPEL4WS**. In *Proc. of the WWW (Alternate Paper Tracks)*, pages 27–28, 2003.

[42] Carine Courbis and Anthony Finkelstein. **Weaving aspects into web service orchestrations**. In *Proc. of the IEEE Int. Conf. on Web Services ICWS*, pages 219–226. IEEE, 2005.

[43] GR Gangadharan. *Service Licensing*. PhD thesis, DISI-University of Trento, Italy, 2008.

[44] Fabio Barbon, Paolo Traverso, Marco Pistore, and Michele Trainotti. **Run-time monitoring of instances and classes of web service compositions.** In *Proc. of the Int. Conf. on Web Services*, pages 63–71. IEEE, 2006.

[45] Heather Kreger. **Web services conceptual architecture (WSCA 1.0).** Technical report, IBM Software Group, 2001.

[46] Shin Nakajima. **Model-checking behavioral specification of BPEL applications.** *Electronic Notes in Theoretical Computer Sci.*, **151(2)**:89–105, 2006.

[47] Petr Hnětynka and František Plášil. **Dynamic reconfiguration and access to services in hierarchical component models.** In *Component-Based Software Engineering*, pages 352–359. Springer, 2006.

[48] Axel Martens. **Analyzing web service based business processes.** In *Proceedings of the Int. Conf. on Fundamental Approaches to Software Engineering*, pages 19–33. Springer, 2005.

[49] Rajeev Alur, Thomas A Henzinger, and Orna Kupferman. **Alternating-time temporal logic.** *Journal of the ACM (JACM)*, **49**(5):672–713, 2002.

[50] Holger Schlingloff, Axel Martens, and Karsten Schmidt. **Modeling and Model Checking Web Services.** *Electronic Notes in Theoretical Computer Sci.*, **126**:3–26, 2005.

[51] Pearl Brereton, Barbara A Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. **Lessons from applying the systematic literature review process within the software Eng. domain.** *Journal of systems and software*, **80(4)**:571–583, 2007.

[52] Pooyan Jamshidi, Aakash Ahmad, and Claus Pahl. **Cloud migration research: a systematic review.** *IEEE Trans. on Cloud Computing*, **1**(2):142–157, 2013.

[53] Chandrashekar Jatoth, GR Gangadharan, and Rajkumar Buyya. **Computational intelligence based QoS-aware web service composition: A systematic literature review.** *IEEE Trans. on Services Computing*, **10**:475–492, 2017.

[54] GR Gangadharan, Vincenzo D'Andrea, and Michael Weiss. **Service Licensing Composition and Compatibility Analysis.** *International Journal of Cooperative Information Systems*, **17**(03):301–317, 2008.

[55] Hai Huang, W-T Tsai, and Raymond Paul. **Proof slicing with application to model checking web services.** In *Proc. of the 8th Int. Symp. on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pages 292–299. IEEE, 2005.

[56] Guilan Dai, Xiaoying Bai, and Chongchong Zhao. **A framework for model checking web service compositions based on BPEL4WS.** In *Proc. of the Int. Conf. on e-Business Engineering*, pages 165–172. IEEE, 2007.

[57] Jian-yin Zhang, Fang-chun Yang, and SU Sen. **Detecting feature interactions in web services with model checking techniques.** *The Journal of China Universities of Posts and Telecommunications*, **14**(3):108–112, 2007.

[58] Flavio Corredini, AP De Angelis, and Alberto Polzonetti. **Verification of WS-CDL choreographies.** In *Proc. of the 2nd European Young Researchers Workshop on Service Oriented Computing (YR-SOC 2007)*, pages 13–18, 2007.

[59] Junyan Qian, Guoyong Cai, Tianlong Gu, and Lingzhong Zhao. **Abstract model checking for Web services.** *Wuhan University Journal of Natural Sci.*, **13**(4):466–470, 2008.

[60] Rongsheng Dong, Zhao Wei, and Xiangyu Luo. **Model checking behavioral specification of BPEL web services.** In *Proc. of the World Congr. on Engineering*, **1**, 2008.

[61] Rujuan Liu, Changjun Hu, and Chongchong Zhao. **Model Checking for Web Service Flow Based on Annotated OWL-S.** In *Proc. of the 9th ACIS Int. Conf. on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, pages 741–746, 2008.

[62] Xiangyu Luo, Zheng Tan, and Rongsheng Dong. **Automatic Verification of Composite Web Services Based on Temporal and Epistemic Logic.** In *Proc. of the 3rd Int. Conf. on Genetic and Evolutionary Computing*, pages 693–696. IEEE, 2009.

[63] Adam Barker, Christopher D Walton, and David Robertson. **Choreographing web services**. *IEEE Trans. on Services Computing*, **2**(2):152–166, 2009.

[64] Dong Xu, Zhou Lei, Wei-min Li, and Bo-feng Zhang. **Model checking web services choreography in process analysis toolkit**. *Journal of Shanghai University (English Edition)*, **14**:45–49, 2010.

[65] Xiangyu Luo, Jingjing Lu, Kaile Su, and Rongsheng Dong. **Translation-based verification of Web services composition via ZING**. In *Proc. of the Int. Conf. on Intelligent Computing and Integrated Systems (ICISS)*, pages 596–600. IEEE, 2010.

[66] Jia Mei, Huaikou Miao, Qingguo Xu, and Pan Liu. **Modeling and verifying web service applications with time constraints**. In *Proc. of the 9th Int. Conf. on Computer and Inform. Sci. (ICIS)*, pages 791–795. IEEE, 2010.

[67] Zhang Xinlin. **Categorical Description and Checking for Web Services Composition Model**. In *Proc. of the 4th Int. Symp. on Computational Intelligence and Design (ISCID)*, pages 206–210. IEEE, 2011.

[68] Guangquan Zhang, Huijuan Shi, Mei Rong, and Haojun Di. **Model checking for asynchronous web service composition based on XYZ/ADL**. In *Proc. of the Int. Conf. on Web Inform. Systems and Mining*, pages 428–435. Springer, 2011.

[69] Wenrui Li, Zhongxue Yang, Pengcheng Zhang, and Zhijian Wang. **Model checking WS-BPEL with universal modal sequence diagrams**. In *Proc. of the 10th Int. Conf. on Computer and Inform. Sci. (ICIS)*, pages 328–333. IEEE, 2011.

[70] Dongwei Fu and Guobin Chen. **A verification method for web services combination based on abstract and refinement technology**. In *Proc. of the 2nd Int. Conf. on Consumer Electronics, Communications and Networks (CECNet)*, pages 119–122. IEEE, 2012.

[71] Xiangyu Luo, Kun Wang, and Fengchai Wang. **An Epistemic Model Checking Approach to Web Service Compositions**. *Int. Journal of Wireless and Microwave Technologies (IJWMT)*, **2**(6):66, 2012.

[72] Valeriu Todica, M-F Vaida, and Marcel Cremene. **Formal verification in Web services composition**. In *Proc. of the IEEE Int. Conf. on Automation Quality and Testing Robotics (AQTR)*, pages 195–200. IEEE, 2012.

[73] Huiqun Zhao, Jing Sun, and Xiaodong Liu. **A model checking based approach to automatic test suite generation for testing web services and BPEL**. In *Proc. of the Asia-Pacific Services Computing Conf. (APSCC)*, pages 61–69. IEEE, 2012.

[74] Hai Huang, W-T Tsai, and Raymond Paul. **Automated Model Checking and Testing for Composite Web Services**. In *Proc. of the 8th IEEE Int. Symp. on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pages 300–307, 2005.

[75] Natasha Sharygina and Daniel Kröning. **Model checking with abstraction for web services**. In *Test and Analysis of Web Services*, pages 121–145. Springer, 2007.

[76] Jia Mei, Huaikou Miao, Yihai Chen, and Honghao Gao. **Verifying Web Services Composition Based on Interface Automata Using SPIN**. *Int. Journal of Digital Content Technology and its Applications*, **4**(8):23–33, 2010.

[77] Huiqun Zhao, Wenwen Wang, Jing Sun, and Ying Wei. **Research on formal modeling and verification of BPEL-based Web service composition**. In *Proc. of the 11th Int. Conf. on Computer and Inform. Sci. (ICIS)*, pages 631–636. IEEE, 2012.

[78] Fang Yu, Chao Wang, Aarti Gupta, and Tevfik Bultan. **Modular verification of web services using efficient symbolic encoding and summarization**. In *Proc. of the 16th ACM SIGSOFT Int. Symp. on Foundations of software engineering*, pages 192–202. ACM, 2008.

[79] Gregorio Diaz, María-Emilia Cambronero, Juan José Pardo, Valentin Valero, and Fernando Cuartero. **Automatic generation of correct web services choreographies and orchestrations with model checking techniques.** In *Proc. of the Advanced Int'l Conf. on Telecommunications and Int'l Conf. on Internet and Web Applications and Services (AICT-ICIW'06)*, pages 186–186. IEEE, 2006.

[80] Wing Lok Yeung. **Mapping WS-CDL and BPEL into CSP for Behavioural Specification and Verification of Web Services.** In *Proc. of the European Conf. on Web Services (ECOWS)*, **6**, pages 297–305, 2006.

[81] Hongbing Wang, Li Li, Chen Wang, Zuling Kang, Dongxi Liu, Jemma Wu, and Athman Bouguettaya. **Logic-based verification for Web services composition with TLA.** In *Proc. of the Int. Conf. on Service-Oriented Computing and Applications (SOCA)*, pages 1–8. IEEE, 2009.

[82] Sylvain Hallé. **Automated generation of web service stubs using LTL satisfiability solving.** In *Proc. of the Int. Workshop on Web Services and Formal Methods*, pages 42–55. Springer, 2010.

[83] Lina Bentakouk, Pascal Poizat, and Fatiha Zaïdi. **Checking the behavioral conformance of web services with symbolic testing and an SMT solver.** In *Proc. of the Int. Conf. on Tests and Proofs*, pages 33–50. Springer, 2011.

[84] Honghao Gao, Yuanyuan Zhang, and Jun Liu. **Modeling and verifying web service behaviors based on live sequence chart specifications [j].** *Journal of Computational Inform. Systems*, **7**(10):3499–3507, 2011.

[85] Gregorio Díaz, M Emilia Cambronero, Hermenegilda Maciá, and Valentín Valero. **Model-checking verification of publish-subscribe architectures in web service contexts.** In *Proc. of the 30th Annual ACM Symp. on Applied Computing*, pages 1688–1695. ACM, 2015.

[86] Giti Oghabi, Jamal Bentahar, and Abdelghani Benharref. **Model Checking Single Web Services using Markov Chains and MDPs.** In *Proc. of the*

New Trends in Software Methodologies, Tools and Techniques (SoMeT), pages 20–37, 2011.

[87] Arbia Ben Azaiez and Zohra Sbaï. **Model Checking Web Services Choreography**. In *Proc. of the 11th Int. Workshop on Enterprise and Organizational Modeling and Simulation, EOMAS*, pages 171–186. Springer, 2015.

[88] Anders P Ravn, Jiří Srba, and Saleem Vighio. **Modelling and verification of web services business activity protocol**. In *Proc. of the Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, pages 357–371. Springer, 2011.

[89] Giti Oghabi, Jamal Bentahar, and Abdelghani Benharref. **On the verification of behavioral and probabilistic web services using transformation**. In *Proc. of the Int. Conf. on Web Services (ICWS)*, pages 548–555. IEEE, 2011.

[90] Jr. Marques, AbinoamP., AndersP. Ravn, Jia Srba, and Saleem Vighio. **Model-checking web services business activity protocols**. *Int. Journal on Software Tools for Technology Transfer*, **15(2)**:125–147, 2013.

[91] Howard Foster, Sebastian Uchitel, Jeff Magee, and Jeff Kramer. **Model-based verification of web service compositions**. In *Proc. of the 18th IEEE Int. Conf. on Automated Software Engineering*, pages 152–161. IEEE, 2003.

[92] Xiang Fu, Tevfik Bultan, and Jianwen Su. **Analysis of Interacting BPEL Web Services**. In *Proc. of the 13th Int. Conf. on World Wide Web*, WWW '04, pages 621–630. ACM, 2004.

[93] Raman Kazhamiakin, Marco Pistore, and Marco Roveri. **Formal verification of requirements using spin: A case study on web services**. In *Proc. of the 2nd Int. Conf. on Software Eng. and Formal Methods*, pages 406–415. IEEE, 2004.

[94] Christopher Walton. **Model checking multi-agent web services**. In *Proc. of the AAAI Symp. on Semantic Web Services*, 2004.

[95] Xiang Fu, Tevfik Bultan, and Jianwen Su. **Realizability of conversation protocols with message contents**. In *Proc. of the Int. Conf. on Web Services*, pages 96–103. IEEE, 2004.

[96] Marco Pistore, Marco Roveri, and Paolo Busetta. **Requirements-driven verification of web services**. In *Proc. of the 1st Int. Workshop on Web Services and Formal Methods (WSFM'04)*, pages 95–108. Elsevier, 2004.

[97] Honghua Cao, Shi Ying, and Dehui Du. **Towards model-based verification of BPEL with model checking**. In *Proc. of the 6th IEEE Int. Conf. on Computer and Inform. Technology (CIT'06)*, pages 190–190. IEEE, 2006.

[98] Gregorio Díaz, Juan José Pardo, María-Emilia Cambronero, Valentin Valero, and Fernando Cuartero. **Verification of Web Services with Timed Automata**. *Electr. Notes in Theor. Comput. Sci.*, **157(2)**:19–34, 2006.

[99] Yongyan Zheng, Jiong Zhou, and P. Krause. **A Model Checking based Test Case Generation Framework for Web Services**. In *Proc. of the 4th Int. Conf. on Inform. Technology (ITNG '07)*, pages 715–722. IEEE, 2007.

[100] Alin Deutsch, Liying Sui, and Victor Vianu. **Specification and verification of data-driven web applications**. *Journal of Computer and System Sci.*, **73**(3):442–474, 2007.

[101] Zhifeng Gu, Juanzi Li, Jie Tang, Bin Xu, and Ruobo Huang. **Verification of web service conversations specified in wscl**. In *Proc. of the 31st Int. Computer Software and Applications Conf. (COMPSAC'07)*, **2**, pages 432–437. IEEE, 2007.

[102] Pavel Parizek and Jiri Adamek. **Checking Session-Oriented Interactions between Web Services**. In *Proc. of the 34th Euromicro Conf. Software Eng. and Advanced Applications*, pages 3–10. IEEE, 2008.

[103] German Shegalov and Gerhard Weikum. **Formal verification of web service interaction contracts**. In *Proc. of the IEEE Int. Conf. on Services Computing (SCC'08)*, **2**, pages 525–528. IEEE, 2008.

[104] Wei Wan, Jamal Bentahar, and Abdessamad Ben Hamza. **Modeling and verifying agent-based communities of web services**. In *Proc. of the Int. Conf. on Industrial, Eng. and Other Applications of Applied Intelligent Systems*, pages 418–427. Springer, 2010.

[105] Ling-li Qian and Yi-hai Chen. **Generating test case specifications of web service composition using model checking**. *Journal of Shanghai University (English Edition)*, **15**:409–414, 2011.

[106] Quan Z. Sheng, Zakaria Maamar, Lina Yao, Claudia Szabo, and Scott Bourne. **Behavior modeling and automated verification of Web services**. *Inform. Sci.*, **258**:416–433, 2014.

[107] Alin Deutsch, Liying Sui, Victor Vianu, and Dayou Zhou. **Verification of communicating data-driven web services**. In *Proc. of the 25th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 90–99. ACM, 2006.

[108] Warda El Kholy, Mohamed El Menshawy, Jamal Bentahar, Hongyang Qu, and Rachida Dssouli. **Verifying multiagent-based web service compositions regulated by commitment protocols**. In *Proc. of the IEEE Int. Conf. on Web Services (ICWS)*, pages 49–56. IEEE, 2014.

[109] Majdi Ghannoudi and Walid Chainbi. **Formal verification for Web service composition: A model-checking approach**. In *Proc. of the Int. Symp. on Networks, Computers and Communications (ISNCC)*, pages 1–6. IEEE, 2015.

[110] Lucas Bordeaux, Gwen Salaün, Daniela Berardi, and Massimo Mecella. **When are two web services compatible?** In *Technologies for E-Services*, pages 15–28. Springer, 2004.

[111] Jyotishman Pathak, Samik Basu, and Vasant Honavar. **On context-specific substitutability of web services**. In *Proc. of the IEEE Int. Conf. on Web Services (ICWS)*, pages 192–199. IEEE, 2007.

[112] Hai Dong, Farookh Khadeer Hussain, and Elizabeth Chang. **Semantic Web Service matchmakers: state of the art and challenges**. *Concurrency and Computation: Practice and Experience*, **25**(**7**):961–988, 2013.

[113] Mingwei Zhang, Chengfei Liu, Jian Yu, Zhiliang Zhu, and Bin Zhang. **A correlation context-aware approach for composite service selection**. *Concurrency and Computation: Practice and Experience*, **25**(**13**):1909–1927, 2013.

[114] Axel Martens. **On compatibility of web services**. *Petri Net Newsletter*, **65**(12-20):100, 2003.

[115] Andrea Bracciali, Antonio Brogi, and Carlos Canal. **A formal approach to component adaptation**. *Journal of Systems and Software*, **74**(**1**):45–54, 2005.

[116] Li Kuang, Yingjie Xia, Shuiguang Deng, and Jian Wu. **Analyzing behavioral substitution of web services based on pi-calculus**. In *Proc. of the IEEE Int. Conf. on Web Services (ICWS)*, pages 441–448. IEEE, 2010.

[117] Fangfang Liu, Yuliang Shi, Liang Zhang, Lili Lin, and Baile Shi. **Analysis of web services composition and substitution via CCS**. In *Data Eng. Issues in E-Commerce and Services*, pages 236–245. Springer, 2006.

[118] Andreas Both and Wolf Zimmermann. **Automatic protocol conformance checking of recursive and parallel BPEL systems**. In *Proc. of the IEEE 6th European Conf. on Web Services, ECOWS'08.*, pages 81–91. IEEE, 2008.

[119] Aziz Salah, Guy Tremblay, and Aida Chami. **Behavioral interface conformance checking for ws-bpel processes**. In *Proc. of the Int. MCETECH Conf. on e-Technologies*, pages 253–257. IEEE, 2008.

[120] Akin Günay and Pinar Yolum. **Semantic matchmaking of web services using model checking**. In *Proc. of the 7th Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, pages 273–280, 2008.

[121] Alessio Lomuscio, Hongyang Qu, and Monika Solanki. **Towards verifying compliance in agent-based web service compositions**. In *Proc. of the 7th Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, pages 265–272, 2008.

[122] Nawal Guermouche and Claude Godart. **Timed model checking based approach for web services analysis**. In *Proc. of the IEEE Int. Conf. on Web Services, ICWS*, pages 213–221. IEEE, 2009.

[123] Marcello M Bersani, Luca Cavallaro, Achille Frigeri, Matteo Pradella, and Matteo Rossi. **SMT-based verification of LTL specification with integer constraints and its application to runtime checking of service substitutability**. In *Proc. of the 8th Int. Conf. on Software Eng. and Formal Methods (SEFM)*, pages 244–254. IEEE, 2010.

[124] Wing Lok Yeung. **A formal and visual modeling approach to choreography based web services composition and conformance verification**. *Expert Systems with Applications*, **38**(10):12772–12785, 2011.

[125] Javier Cámara, Gwen Salaün, Carlos Canal, and Meriem Ouederni. **Interactive specification and verification of behavioral adaptation contracts**. *Information and Software Technology*, **54**(7):701–723, 2012.

[126] H. Groefsema, N. van Beest, and M. Aiello. **A Formal Model for Compliance Verification of Service Compositions**. *IEEE Trans. on Services Computing*, **10.1109/TSC.2016.2579621**, 2016.

[127] Khai Huynh, Tho Quan, and Thang Bui. **Smaller to sharper: efficient web service composition and verification using on-the-fly model checking and logic-based clustering**. In *Proc. of the Int. Conf. on Computational Sci. and Its Applications*, pages 453–468. Springer, 2016.

[128] Ahmed Saleh Bataineh, Jamal Bentahar, Mohamed El Menshawy, and Rachida Dssouli. **Specifying and Verifying Contract-driven Service Compositions using Commitments and Model Checking**. *Expert Systems with Applications*, **74**:151 – 184, 2016.

[129] Zohra Sbaï and Rawand Guerfel. **CTL Model Checking of Web Services Composition based on Open Workflow Nets Modeling**. *Int. Journal of Service Science, Management, Engineering, and Technology (IJSSMET)*, 7(1):27–42, 2016.

[130] Shin Nakajima. **Model-checking of safety and security aspects in web service flows**. In *Proc. of the Int. Conf. on Web Engineering*, pages 488–501. Springer, 2004.

[131] Marina Mongiello and Daniela Castelluccia. **Modelling and verification of BPEL business processes**. In *Proc. of the 4th Workshop on Model-Based Development of Computer-Based Systems and 3rd Int. Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MBD-MOMPES'06)*, pages 5–9. IEEE, 2006.

[132] Zhao Xiangpeng, Antonio Cerone, and Padmanabhan Krishnan. **Verifying BPEL workflows under authorisation constraints**. In *Proc. of the Int. Conf. on Business Process Management*, pages 439–444. Springer, 2006.

[133] Ye Xiaolie and Liao Lejian. **Verifying a Secure Session Protocol for Web Services**. In *Proc. of the Int. Conf. on Networks Security, Wireless Communications and Trusted Computing, (NSWCTC'09)*, 2, pages 301–304. IEEE, 2009.

[134] Guangquan Zhang, Mei Rong, Yali He, Xueyang Zhu, and Rongjie Yan. **A refinement checking method of web services composition**. In *Proc. of the 5th IEEE Int. Symp. on Service Oriented System Eng. (SOSE)*, pages 103–106. IEEE, 2010.

[135] Honghao Gao, Huaikou Miao, and Hongwei Zeng. **Predictive web service monitoring using probabilistic model checking**. *Appl. Math*, 7(1L):139–148, 2013.

[136] Jiajun Lu, Zhiqiu Huang, and Changbo Ke. **Verification of Behavior-aware Privacy Requirements in Web Services Composition**. *Journal of Software*, 9(4):944–951, 2014.

[137] Hu Jingjing, Zhu Wei, Zhao Xing, and Zhu Dongfeng. **Web Service Composition Automation based on Timed Automata**. *Applied Mathematics & Inform. Sci.*, **8**(4):2017–2024, July 2014.

[138] Zhang Rui-Min and Li Xiao-Bin. **Model Checking of Non-Centralized Automaton Web Service with AMT Bounded Constraint**. *Int. Journal of Multimedia and Ubiquitous Eng.*, **11**(3):57–66, 2016.

[139] Chengyang Mi, Huaikou Miao, Jinyu Kai, and Honghao Gao. **Reliability modeling and verification of BPEL-based web services composition by probabilistic model checking**. In *Proc. of the 14th Int. Conf. on Software Eng. Research, Management and Applications (SERA)*, pages 149–154. IEEE, 2016.

[140] Jumana El-Qurna, Hamdi Yahyaoui, and Mohamed Almulla. **A New Framework for the Verification of Service Trust Behaviors**. *Knowledge-Based Systems*, 2017.

[141] Gregorio Díaz, María-Emilia Cambronero, M Llanos Tobarra, Valentín Valero, and Fernando Cuartero. **Analysis and verification of time requirements applied to the web services composition**. In *Proc. of the Int. Workshop on Web Services and Formal Methods*, pages 178–192. Springer, 2006.

[142] R. Kazhamiakin, P. Pandya, and M. Pistore. **Timed modelling and analysis in Web service compositions**. In *Proc. of the 1st Int. Conf. on Availability, Reliability and Security (ARES'06)*, pages 7 pp.–846. IEEE, 2006.

[143] Enrique Martinez, Maria Emilia Cambronero, Gregorio Diaz, and Valentin Valero. **Design and verification of web services compositions**. In *Proc. of the 4th Int. Conf. on Internet and Web Applications and Services (ICIW'09)*, pages 395–400. IEEE, 2009.

[144] Guerfel Rawand, Zohra Sbaï, and Kamel Barkaoui. **Modeling and formal verification framework of Web services composition**. In *Proc. of the Int. Conf. on Control, Eng. Inform. Technology (CEIT'13)*, **2**, pages 140–145, 2013.

[145] Shin Nakajima. **Verification of web service flows with model-checking techniques**. In *Proc. of the 1st Int. Symp. on Cyber Worlds*, pages 378–385. IEEE, 2002.

[146] Aysu Betin-Can, Tevfik Bultan, and Xiang Fu. **Design for verification for asynchronously communicating web services**. In *Proc. of the 14th Int. Conf. on World Wide Web*, pages 750–759. ACM, 2005.

[147] Zhao Xiangpeng, Yang Hongli, and Qiu Zongyan. **Towards the formal model and verification of web service choreography description language**. In *Proc. of the Int. Workshop on Web Services and Formal Methods*, pages 273–287. Springer, 2006.

[148] G Dìaz and ME Cambronero. **Model Checking Techniques applied to the design of Web Services**. *CLEI Electronic*, 10(2), 2007.

[149] Cátia Vaz and Carla Ferreira. **Towards automated verification of web services**. In *Proc. of the IADIS Int. Conf. on WWW/Internet*, 2007.

[150] Yang Hongli, Zhao Xiangpeng, Cai Chao, and Qiu Zongyan. **Model-checking of web services choreography**. In *Proc. of the IEEE Int. Symp. on Service-Oriented System Eng. (SOSE'08)*, pages 79–84. IEEE, 2008.

[151] Thang Huynh Quyet, Quynh Pham Thi, and Duc Bui Hoang. **A method of verifying web service composition**. In *Proc. of the Symp. on Inform. and Communication Technology*, pages 155–162. ACM, 2010.

[152] Honghao Gao, Huaikou Miao, Shengbo Chen, and Jia Mei. **Probabilistic timed model checking for atomic Web service**. In *Proc. of the IEEE World Congr. on Services*, pages 459–466. IEEE, 2011.

[153] Naseem Ibrahim and Ismail Khalil. **Verifying web services compositions using UPPAAL**. In *Proc. of the Int. Conf. on Computer Systems and Industrial Informatics (ICCSII)*, pages 1–5. IEEE, 2012.

[154] Khai T Huynh, Tho T Quan, and Thang H Bui. **A bitwise-based indexing and heuristic-driven on-the-fly approach for Web service composition and verification**. *Vietnam Journal of Computer Science*, pages 1–16, 2016.

[155] Alessio Lomuscio, Hongyang Qu, Marek Sergot, and Monika Solanki. **Verifying temporal and epistemic properties of web service compositions**. In *Proc. of the Int. Conf. on Service-Oriented Computing*, pages 456–461. Springer, 2007.

[156] Sebastian Wieczorek, Vitaly Kozyura, Andreas Roth, Michael Leuschel, Jens Bendisposto, Daniel Plagge, and Ina Schieferdecker. **Applying model checking to generate model-based integration tests from choreography models**. In *Testing of Software and Communication Systems*, pages 179–194. Springer, 2009.

[157] Liyang Peng, Chao Cai, Qiu Zongyan, and Geguang Pu. **Verification of channel passing in choreography with model checking**. In *Proc. of the Int. Conf. on Service-Oriented Computing and Applications (SOCA)*, pages 1–5. IEEE, 2009.

[158] Zuling Kang and Hongbing Wang. **Verifying WS-CDL-Based Web Services Collaboration by Model Checking**. In *Proc. of the Congr. on Services-I*, pages 554–561. IEEE, 2009.

[159] Zhi Fang, Lejian Liao, and Ruoyu Chen. **Bounded Model Checking for Web Service Discovery and Composition**. In *Proc. of the Software Eng. Artificial Intelligence Networking and Parallel/Distributed Computing (SNPD)*, pages 201–206. IEEE, 2010.

[160] Sabina Rossi. **Model Checking Adaptive Multilevel Service Compositions**. In *Proc. of the 7th Int. Workshop on Formal Aspects of Component Software, FACS*, pages 106–124, 2010.

[161] Alessio Lomuscio, Hongyang Qu, and Monika Solanki. **Towards verifying contract regulated service composition**. *Autonomous Agents and Multi-Agent Systems*, **24(3)**:345–373, 2012.

[162] Li Yongxiang, Yao Xifan, Zhang Jie, and Li Bin. **Cloud manufacturing service composition modeling and formal verification based on Calculus for Orchestration of Web Service**. In *Proc. of the 25th Chinese Control and Decision Conf. (CCDC)*, pages 2806–2810. IEEE, 2013.

[163] Yeon-Seok Kim, Dong-Hoon Shin, Hyun-Bae Jeon, Kyong-Ho Lee, Kee-Seong Cho, and Wonjoo Park. **Conflict detection in composite web services based on model checking**. *Int. Journal of Web and Grid Services*, 9(4):394–430, 2013.

[164] Natallia Kokash and Farhad Arbab. **Formal design and verification of long-running transactions with extensible coordination tools**. *IEEE Trans. on Services Computing*, **6(2)**:186–200, 2013.

[165] Scott Bourne, Claudia Szabo, and Quan Z. Sheng. **Verifying Transactional Requirements of Web Service Compositions Using Temporal Logic Templates**. In *Proc. of the 14th Int. Conf. on Web Information Systems Engineering WISE*, pages 243–256, 2013.

[166] Frank Leymann. *Web services flow language (WSFL 1.0)*. IBM Academy of Technology, 2001.

[167] A. Airkin, S. Askary, S. Fordin, W. Jekeli, D. Orchard, and K. Riemer. **Web Service Choreography Interface WSCI 1.0**, 2002.

[168] Gerard J. Holzmann. *The SPIN model checker: Primer and reference manual*, **1003**. Addison-Wesley Reading, 2004.

[169] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. **Nusmv 2: An opensource tool for symbolic model checking**. In *Proc. of the Computer Aided Verification*, pages 359–364. Springer, 2002.

[170] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. **UPPAAL — a Tool Suite for Automatic Verification of Real-Time Systems**. In *Proc. of the Workshop on Verification and Control of Hybrid Systems III*, pages 232–243. Springer, 1995.

[171] Luca De Alfaro, Marta Kwiatkowska, Gethin Norman, David Parker, and Roberto Segala. **Symbolic model checking of probabilistic processes using MTBDDs and the Kronecker representation**. In *Proc. of the Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, pages 395–410. Springer, 2000.

[172] Sjoerd Cranen, Jan Friso Groote, Jeroen JA Keiren, Frank PM Stappers, Erik P de Vink, Wieger Wesselink, and Tim AC Willemse. **An overview of the mCRL2 toolset and its recent advances**. In *Proc. of the Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, pages 199–213. Springer, 2013.

[173] Yang Liu, Jun Sun, and Jin Song Dong. **Pat 3: An extensible architecture for building multi-domain model checkers**. In *Proc. of the 22nd Int. Symp. on Software Reliability Eng. (ISSRE)*, pages 190–199. IEEE, 2011.

[174] Tony Andrews, Shaz Qadeer, Sriram K Rajamani, Jakob Rehof, and Yichen Xie. **Zing: Exploiting program structure for model checking concurrent software**. In *Proc. of the Int. Conf. on Concurrency Theory*, pages 1–15. Springer, 2004.

[175] Dirk Beyer, Thomas A Henzinger, Ranjit Jhala, and Rupak Majumdar. **The software model checker Blast**. *Int. Journal on Software Tools for Technology Transfer*, 9(5-6):505–525, 2007.

[176] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. **MCMAS: A model checker for the verification of multi-agent systems**. In *Proc. of the Int. Conf. on Computer Aided Verification*, pages 682–688. Springer, 2009.

[177] Maribel Fernández. *Models of Computation: An Introduction to Computability Theory*. Springer Sci. & Business Media, 2009.

[178] T. A. Sudkamp, editor. *Languages and Machines : An Introduction to the Theory of Computer Science*. Addison-Wesley Longman Publishing Co., Inc., 1988.

[179] Rachid Hamadi and Boualem Benatallah. **A Petri net-based model for web service composition**. In *Proc. of the 14th Australian Database Conf.*, pages 191–200. Australian Computer Society, Inc., 2003.

[180] SV Hashemian and F Mavaddat. **Composition algebra**. In *Proc. of the Formal Aspects of Component Software (FACS'06)*, pages 247–264, 2006.

[181] Peter Höfner and Florian Lautenbacher. **Algebraic structure of web services**. *Electronic Notes in Theoretical Computer Science*, **200**(3):171–187, 2008.

[182] Qiang Hu, Yuyue Du, and ShuXia Yu. **Service net algebra based on logic Petri nets**. *Information Sci.*, **268**:271–289, 2014.

[183] Qi Yu and Athman Bouguettaya. **Framework for web service query algebra and optimization**. *ACM Trans. on the Web (TWEB)*, **2**(1):6, 2008.

[184] Andrea Ferrara. **Web services: a process algebra approach**. In *Proc. of the 2nd Int. Conf. on Service Oriented Computing*, pages 242–251. ACM, 2004.

[185] Petros Papapanagiotou and Jacques D. Fleuriot. **A theorem proving framework for the formal verification of Web Services Composition**. In *Proc. of the 7th Int. Workshop on Automated Specification and Verification of Web Systems, WWV*, pages 1–16, 2011.

[186] Hamid Reza Motahari Nezhad, Guang Yuan Xu, and Boualem Benatallah. **Protocol-aware matching of web service interfaces for adapter development**. In *Proc. of the 19th Int. Conf. on World Wide Web*, pages 731–740. ACM, 2010.

[187] Pierluigi Plebani and Barbara Pernici. **URBE: Web service retrieval based on similarity evaluation**. *IEEE Trans. on Knowledge and Data Engineering*, **21**(11):1629–1642, 2009.

[188] Jian Wu and Zhaohui Wu. **Similarity-based web service matchmaking**. In *Proc. of the IEEE Int. Conf. on Services Computing*, pages 287–294. IEEE, 2005.

[189] Matthias Klusch, Benedikt Fries, and Katia Sycara. **Automated semantic web service discovery with OWLS-MX**. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 915–922. ACM, 2006.

[190] Jan A Bergstra, Alban Ponse, and Scott A Smolka. *Handbook of process algebra*. Elsevier, 2001.

[191] Gwen Salaun, Lucas Bordeaux, and Marco Schaerf. **Describing and reasoning on web services using process algebra**. *International Journal of Business Process Integration and Management*, **1**(2):116–128, 2006.

[192] Meriem Ouederni and Gwen Salaün. **Tau Be or not Tau Be? - A Perspective on Service Compatibility and Substitutability**. In *Proc. of the Int. Workshop on Component and Service Interoperability, WCSI*, pages 57–70, 2010.

[193] Manman Chen, Tian Huat Tan, Jun Sun, Yang Liu, Jun Pang, and Xiaohong Li. **Verification of functional and non-functional requirements of web service composition**. In *Proc. of the Int. Conf. on Formal Engineering Methods*, pages 313–328. Springer, 2013.

[194] A Prasad Sistla. **Safety, liveness and fairness in temporal logic**. *Formal Aspects of Computing*, **6**(5):495–511, 1994.

[195] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North, and Gordon Woodhull. **Graphviz–open source graph drawing tools**. In *Proc. of the Int. Symp. on Graph Drawing*, pages 483–484. Springer, 2001.

[196] Eirini Kaldeli, Alexander Lazovik, and Marco Aiello. **Domain-independent planning for services in uncertain and dynamic environments**. *Artificial Intelligence*, **236**:30–64, 2016.

[197] Guobing Zou, Yanglan Gan, Yixin Chen, and Bofeng Zhang. **Dynamic composition of Web services using efficient planners in large-scale service repository**. *Knowledge-Based Systems*, **62**:98–112, 2014.

[198] Vimmi Jaiswal, Amit Sharma, and Akshat Verma. **ReComp: QoS-aware recursive service composition at minimum cost.** In *Proc. of the 12th IFIP/IEEE Int. Symp. on Integrated Network Management (IM 2011)*, pages 225–232. IEEE, 2011.

[199] Anissa Abrougui, Annabelle Mercier, Michel Occello, and Marc-Philippe Huget. **Recursive Multi-agent System for Dynamic and Adaptative Web Services Composition**. In *Proc. of the Int. Conf. on Management of Emergent Digital EcoSystems*, pages 44:295–44:299. ACM, 2009.

[200] Hongli Yang, Chao Cai, Liyang Peng, Xiangpeng Zhao, Zongyan Qiu, and Shengchao Qin. **Algorithms for checking channel passing in web service choreography**. *Frontiers of Computer Science*, **7**(5):710–728, 2013.

[201] Achim D Brucker, Francesco Malmignati, Madjid Merabti, Qi Shi, and Bo Zhou. **The Aniketos Service Composition Framework**. In *Secure and Trustworthy Service Composition*, pages 121–135. Springer, 2014.

[202] Agostino Dovier and Carla Piazza. **The Subgraph Bisimulation Problem**. *IEEE Trans. Knowl. Data Eng.*, **15**(4):1055–1056, 2003.

[203] Arthur HM ter Hofstede, Wil van der Aalst, Michael Adams, and Nick Russell. *Modern Business Process Automation: YAWL and its support environment*. Springer Sci. & Business Media, 2009.

[204] Filippo Bonchi, Antonio Brogi, Sara Corfini, and Fabio Gadducci. **A behavioural congruence for web services**. In *Proc. of the Int. Conf. on Fundamentals of Software Engineering*, pages 240–256. Springer, 2007.

[205] Seyyed Vahid Hashemian. *Automatic Signature Matching in Component Composition*. PhD thesis, University of Waterloo, 2008.

[206] Kais Klai, Hanen Ochi, and Samir Tata. **Formal abstraction and compatibility checking of Web services**. In *Proc. of the 20th Int. Conf. on Web Services (ICWS)*, pages 163–170. IEEE, 2013.

[207] Juan Carlos Corrales, Daniela Grigori, Mokrane Bouzeghoub, and Javier Ernesto Burbano. **BeMatch: a platform for matchmaking service behavior models**. In *Proc. of the 11th Int. Conf. on Extending database technology: Advances in database technology*, pages 695–699. ACM, 2008.

[208] Ganesh Ram Santhanam, Samik Basu, and Vasant Honavar. **Web service substitution based on preferences over non-functional attributes**. In *Proc. of the Int. Conf. on Services Computing. SCC'09.*, pages 210–217. IEEE, 2009.

[209] Christian Stahl and Karsten Wolf. **Deciding service composition and substitutability using extended operating guidelines**. *Data & Knowledge Engineering*, **68**(9):819–833, 2009.

[210] Li Kuang, Yuxin Mao, and Yingjie Xia. **A Decidable Formal Analysis of Context-Specific Behavioral Equivalence for Web Services**. *Advanced Sci. Letters*, **4**(4-5):1687–1694, 2011.

[211] Yehia Taher, Djamal Benslimane, Marie-Christine Fauvet, and Zakaria Maamar. **Towards an approach for web services substitution**. In *Proc. of the 10th Int. Database Eng. and Applications Symp., IDEAS'06*, pages 166–173. IEEE, 2006.

[212] Qianhui Liang, Bu-Sung Lee, and Patrick CK Hung. **A rule-based approach for availability of service by automated service substitution**. *Software: Practice and Experience*, **44**(1):47–76, 2014.

[213] Okba Tibermacine, Chouki Tibermacine, and Cherif Foudil. **A Practical Approach to the Measurement of Similarity between WSDL-based Web Services**. In *Proc. of the 6ème Conférence francophone sur les Architectures Logicielles, CAL*, pages 3–18, 2012.

[214] Daniela Grigori, Juan Carlos Corrales, and Mokrane Bouzeghoub. **Behavioral matchmaking for service retrieval**. In *Proc. of the Int. Conf. on Web Services*, pages 145–152. IEEE, 2006.

[215] NVD. `https://nvd.nist.gov/`, Online.

[216] BUGTRAQ. `http://www.securityfocus.com/archive/1`, Online.

[217] XINMING OU AND WAYNE F. BOYER. **A Scalable approach to Attack Graph Generation**. In *Proc. of* 13th *ACM Conf. on Computer and Communications Security (CCS)*, pages 336–345. ACM Press, 2006.

[218] PAUL AMMANN. **Scalable, graph-based network vulnerability analysis**. In *Proc. of the* 9th *ACM Conf. on Computer and Communications Security*, pages 217–224. ACM, 2002.

[219] QINGLIANG CHEN, KAILE SU, CHANJUAN LIU, AND YINYIN XIAO. **Automatic Verification of Web Service Protocols for Epistemic Specifications under Dolev-Yao Model**. In *Proc. of the Int. Conf. on Service Science*, pages 49–54. IEEE, 2010.

[220] S. JAJODIA AND S. NOEL. **Topological Vulnerability Analysis: A Powerful New Approach for Network Attack Prevention, Detection, and Response**. In *Proc. of the Algorithms, Architectures, and Information System Security, Indian Statistical Institute Platinum Jubilee Series*, pages 285–305, 2009.

[221] SVIATOSLAV BRAYNOV AND MURTUZA JADLIWALA. **Representation and analysis of coordinated attacks**. In *Proc. of the ACM workshop on Formal methods in security engineering*, pages 43–51. ACM, 2003.

[222] NESSUS. `http://www.tenable.com/products/nessus`, Online.

[223] RETINA. `http://www.amtsoft.com/retina/`, Online.

[224] GFILANGUARD. `http://www.gfi.com`, Online.

[225] MOHAMMAD GHASEMIGOL, ABBAS GHAEMI-BAFGHI, AND HASSAN TAKABI. **A comprehensive approach for network attack forecasting**. *Computers & Security*, **58**:83–105, 2016.

[226] SHUZHEN WANG, ZONGHUA ZHANG, AND YOUKI KADOBAYASHI. **Exploring attack graph for cost-benefit security hardening: A probabilistic approach**. *Computers & security*, **32**:158–169, 2013.

[227] Jennifer A Cowley, Frank L Greitzer, and Bronwyn Woods. **Effect of network infrastructure factors on information system risk judgments**. *Computers & Security*, **52**:142–158, 2015.

[228] Malik Shahzad Kaleem Awan, Pete Burnap, and Omer Rana. **Identifying cyber risk hotspots: A framework for measuring temporal variance in computer network risk**. *computers & security*, **57**:31–46, 2016.

[229] G. Bopche and B. Mehtre. **Extending Attack Graph-Based Metrics for Enterprise Network Security Management**. In *Proc. of the* 3^{rd} *Int. Conf. on Advanced Computing, Networking and Informatics*, pages 315–325. Springer India, 2016.

[230] Nmap. `https://nmap.org/`, Online.

[231] Dorin Shmaryahu. **Constructing Plan Trees for Simulated Penetration Testing**. In *Proc. of the Doctoral Consortium Dissertation Abstracts, ICPAS*, pages 121–127, 2016.

[232] Meng Sun, Shaodong Li, and Yufei Ou. **Model Checking Business Processes for Web Service Compositions in mCRL2**. In *Proc. of the 6th Int. Conf. on Intelligent Human-Machine Systems and Cybernetics*, 2014.

[233] Anusha Iyer and Hung Q. Ngo. **Towards a Theory of Insider Threat Assessment**. In *Proc. of the IEEE Int. Conf. on Dependable Systems and Networks*, DSN'05, pages 108–117. IEEE, 2005.

[234] Janusz Gorski and Andrzej Wardziński. **Formalising fault trees**. In *Achievement and Assurance of Safety*, pages 311–327. Springer, 1995.

[235] Marc Dacier. *Towards Quantitative Evaluation of Computer Security*. PhD Thesis, Institut National Polytechnique de Toulouse - INPT, 1994.

[236] Marc Dacier and Yves Deswarte. **Privilege Graph: An Extension to the Typed Access Matrix Model**. In *Proc. of the* 3^{rd} *European Symp. on Research in Computer Security*, ESORICS '94, pages 319–334. Springer, 1994.

[237] L.P. Swiler, C. Phillips, D. Ellis, and S. Chakerian. **Computer-attack graph generation tool**. In *Proc. of the DARPA Information Survivability Conf. & Exposition II, DISCEX '01*, **2**, pages 307–321, 2001.

[238] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing. **Automated generation and analysis of attack graphs**. In *Proc. of the IEEE Symp. on Security and Privacy*, pages 273–284, 2002.

[239] S. Jha, O. Sheyner, and J. Wing. **Two Formal Analyses of Attack Graphs**. In *Proc. of the 15th IEEE Workshop on Computer Security Foundations*, CSFW'02, pages 49–63. IEEE, 2002.

[240] Nirnay Ghosh and S.K. Ghosh. **A planner-based approach to generate and analyze minimal attack graph**. *Applied Intelligence*, **36**(2):369–390, 2012.

[241] Barbara Kordy, Ludovic Piètre-Cambacédès, and Patrick Schweitzer. **DAG-based attack and defense modeling: Don't miss the forest for the attack trees**. *Computer Science Review*, **13-14**:1–38, 2014.

[242] J.D. Weiss. **A System Security Engineering Process**. In *Proc. of the* 14^{th} *National Computer Security Conf.*, pages 572–581, 1991.

[243] J. Dawkins, C. Campbell, and J. Hale. **Modeling network attacks: extending the attack tree paradigm**. In *Proc. of the Workshop Statistical Machine Learning Techniques in Computer Intrusion Detection*, 2002.

[244] Andrew Moore, Robert Ellison, and Richard Linger. **Attack Modeling for Information Security and Survivability**. Technical Report CMU/SEI-2001-TN-001, Carnegie Mellon University, USA, 2001.

[245] Indrajit Ray and Nayot Poolsapassit. **Using Attack Trees to Identify Malicious Attacks from Authorized Insiders**. In *Proc. of the* 10^{th} *European Conf. on Research in Computer Security*, ESORICS'05, pages 231–246. Springer, 2005.

[246] B. Schneier. **Attack Trees**. `https://www.schneier.com/paper-attacktrees-ddj-ft.html`.

[247] SecurelTree. **Amenaza Technologies**. `http://www.amenaza.com/`.

[248] Sumeet Jauhar, Binbin Chen, William G. Temple, Xinshu Dong, Zbigniew T. Kalbarczyk, William H. Sanders, and David M. Nicol. **Model-Based Cybersecurity Assessment with NESCOR Smart Grid Failure Scenarios**. In *Proc. of the IEEE* 21st *Pacific Rim Int. Symp. on Dependable Computing (PRDC)*, pages 319–324, 2015.

[249] NESCOR. **Analysis of Selected Electric Sector High Risk Failure Scenarios, Version 1.0**. `http://smartgrid.epri.com/NESCOR.aspx`, September 2013.

[250] R. Ortalo, Y. Deswarte, and M. Kaaniche. **Experimenting with quantitative evaluation tools for monitoring operational security**. *IEEE Trans. on Software Engineering*, **25**(5):633–650, 1999.

[251] Cynthia Phillips and Laura Painton Swiler. **A Graph-based System for Network-vulnerability Analysis**. In *Proc. of the Workshop on New Security Paradigms*, NSPW'98, pages 71–79. ACM, 1998.

[252] Somesh Jha, Oleg Sheyner, and Jeannette M. Wing. **Minimization and Reliability Analyses of Attack Graphs**. Technical report, CMU, USA, 2002.

[253] Steven J. Templeton and Karl Levitt. **A Requires/Provides Model for Computer Attacks**. In *Proc. of the Workshop on New Security Paradigms*, NSPW'00, pages 31–38. ACM, 2001.

[254] Navneet Kumar Pandey, S. K. Gupta, and Shaveta Leekha. **Algebra for Capability Based Attack Correlation**. In *Proc. of the International Workshop on Information Security Theory and Practices WISTP*, pages 117–135, 2008.

[255] Nizar Kheir, Nora Cuppens-Boulahia, Frédéric Cuppens, and Hervé Debar. **A Service Dependency Model for Cost-sensitive Intrusion Response**. In *Proc. of the* 15th *European Conf. on Research in Computer Security*, ESORICS'10, pages 626–642. Springer, 2010.

[256] Steven Noel and Sushil Jajodia. **Managing attack graph complexity through visual hierarchical aggregation**. In *Proc. of the ACM workshop on Visualization and data mining for computer security*, pages 109–118. ACM, 2004.

List of Publications

1. **Gopal N. Rai**, G. R. Gangadharan, and Vineet Padmanabhan. "Algebraic Modeling and Verification of Web Service Composition." In *Proceedings of the 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015)*, London, UK, pp. 675-679, Elsevier, 2015.

2. **Gopal N. Rai**, and G. R. Gangadharan. "Set Partition and Trace Based Verification of Web Service Composition". In *Proceedings of the 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015)*, London, UK, pp. 278-285, Elsevier, 2015.

3. **Gopal N. Rai**, and G. R. Gangadharan. "Architectural Characterization of Web Service Interaction Verification". In *Proceedings of the 3rd International Conference on Advanced Computing, Networking and Informatics (ICACNI)*, Bhubaneswar, India, pp. 447-456, Springer, 2016. [**BEST PAPER AWARD**]

4. **Gopal N. Rai**, G. R. Gangadharan, Vineet Padmanabhan, and Rajkumar Buyya. "Web Service Interaction Modeling and Verification Using Recursive Composition Algebra". Submitted to: *IEEE Transactions on Service Computing*. IEEE. [Under Review]

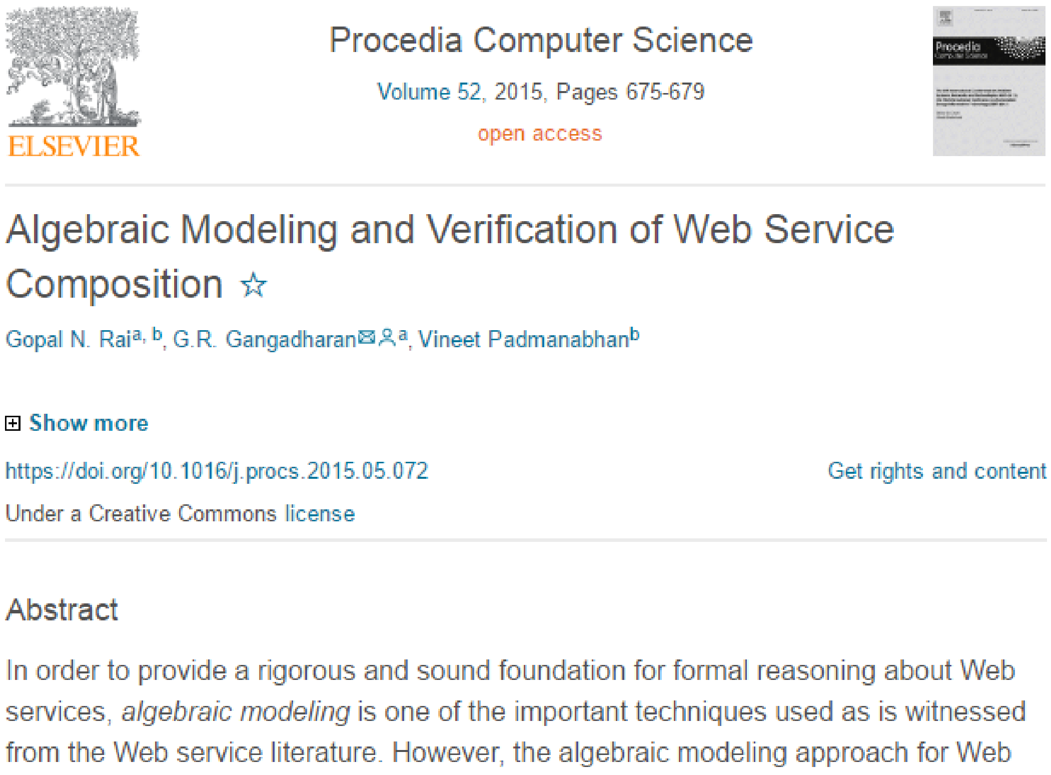
5. **Gopal N. Rai** and G. R. Gangadharan. "Verifying Compositional Equivalence between Web Service Composition Graphs". Submitted to: *Concurrency and Computation: Practice and Experience*. Wiley. [Under Review]

6. **Gopal N. Rai**, Ghanshyam S. Bopche, G. R. Gangadharan, and Babu M. Mehtre. "Modeling and Analyzing Multistage Attacks Using Recursive Composition Algebra". Submitted to: *Computers & Security*. Elsevier. [Under Review]

7. **Gopal N. Rai** and G. R. Gangadharan. "Model Checking Based Web Service Verification: A Systematic Litrerature Review". Submitted to: *IEEE Transactions on Service Computing*. IEEE. [Under Review]

Appendix A

Annexure



Procedia Computer Science

Volume 52, 2015, Pages 675-679

open access

Algebraic Modeling and Verification of Web Service Composition ☆

Gopal N. Rai[a, b], G.R. Gangadharan[a], Vineet Padmanabhan[b]

Show more

https://doi.org/10.1016/j.procs.2015.05.072

Get rights and content

Under a Creative Commons license

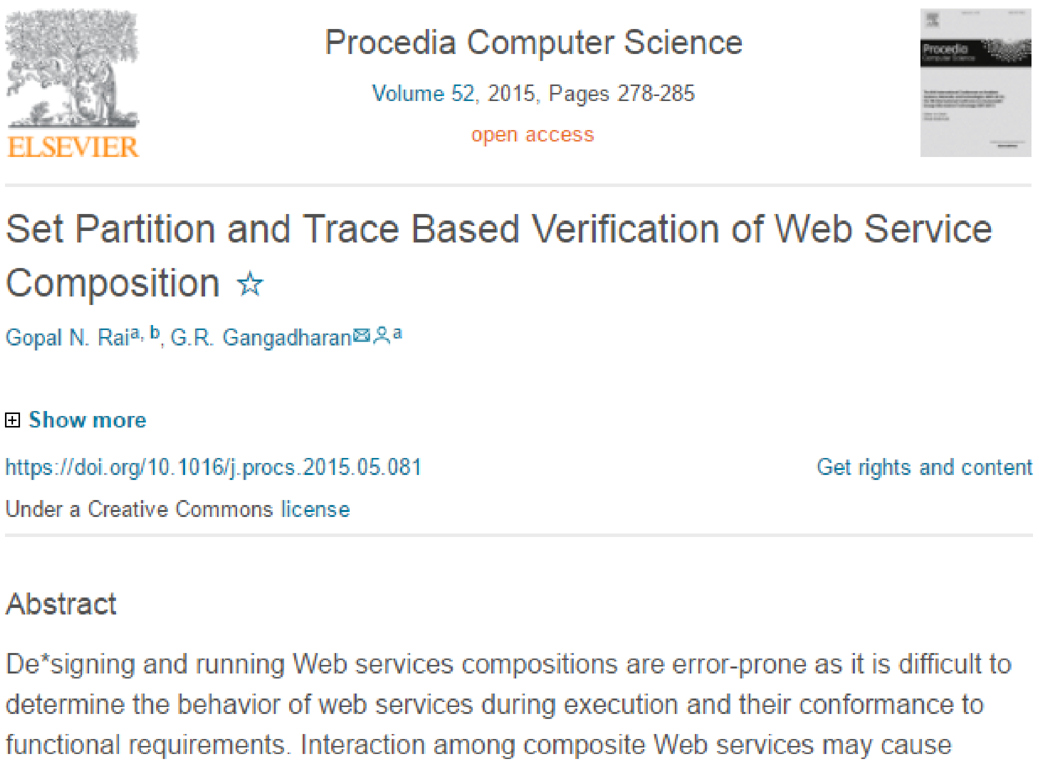
Abstract

In order to provide a rigorous and sound foundation for formal reasoning about Web services, *algebraic modeling* is one of the important techniques used as is witnessed from the Web service literature. However, the algebraic modeling approach for Web

Figure A.1: Publication [2]

Table A.1: Fact Sheet for Publication [2]

Title	Algebraic Modeling and Verification of Web Service Composition
Authors	Gopal N. Rai and G. R. Gangadharan and Vineet Padmanabhan
Publication	In Proc. of the 6th Int. Conf. on Ambient Systems, Networks and Technologies (ANT), 2015, pages 675–679
ISSN	1877-0509
DOI	$10.1016/j.procs.2015.05.072$
Status	Published
Publisher	Elsevier
Publication Type	Conference Proceedings



ELSEVIER

Procedia Computer Science

Volume 52, 2015, Pages 278-285

open access

Set Partition and Trace Based Verification of Web Service Composition ☆

Gopal N. Rai[a, b], G.R. Gangadharan[a]

⊞ Show more

https://doi.org/10.1016/j.procs.2015.05.081

Get rights and content

Under a Creative Commons license

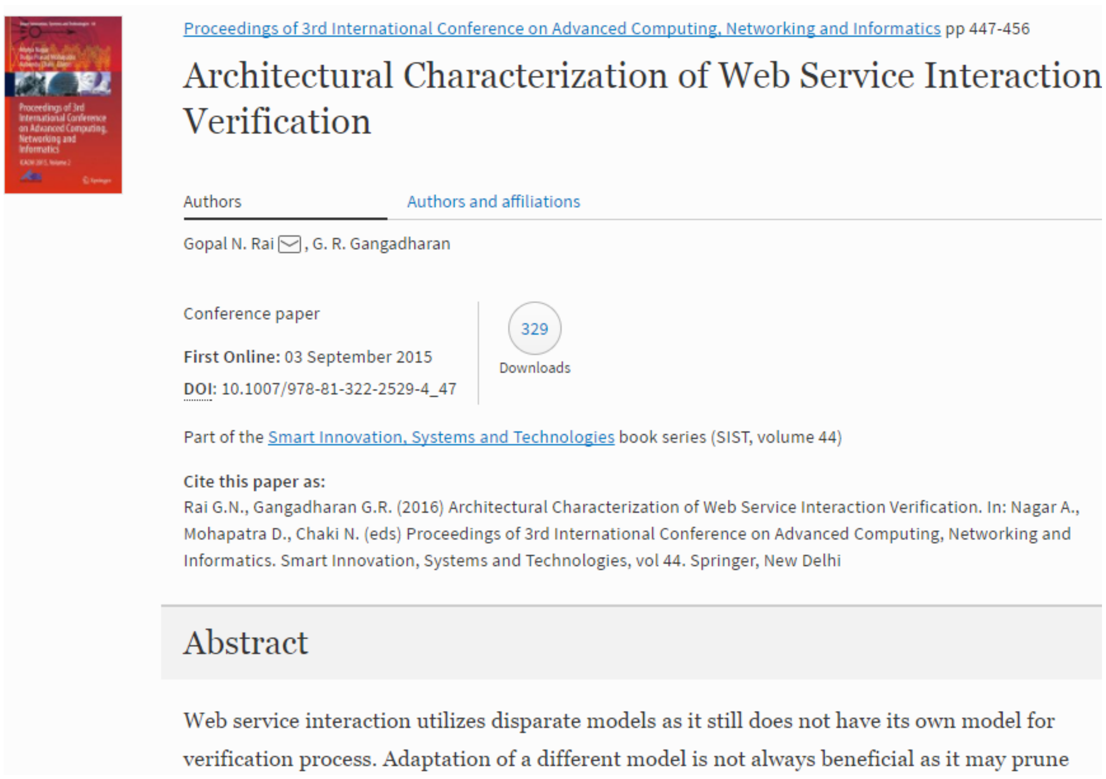
Abstract

De*signing and running Web services compositions are error-prone as it is difficult to determine the behavior of web services during execution and their conformance to functional requirements. Interaction among composite Web services may cause

Figure A.2: Publication [3]

Table A.2: Fact Sheet for Publication [3]

Title	Set Partition and Trace Based Verification of Web Service Composition
Authors	Gopal N. Rai and G. R. Gangadharan
Publication	In Proc. of the 6th Int. Conf. on Ambient Systems, Networks and Technologies (ANT), 2015, pages 278–285
ISSN	1877-0509
DOI	$10.1016/j.procs.2015.05.081$
Status	Published
Publisher	Elsevier
Publication Type	Conference Proceedings



Proceedings of 3rd International Conference on Advanced Computing, Networking and Informatics pp 447-456

Architectural Characterization of Web Service Interaction Verification

Authors | Authors and affiliations

Gopal N. Rai, G. R. Gangadharan

Conference paper

First Online: 03 September 2015

DOI: 10.1007/978-81-322-2529-4_47

329 Downloads

Part of the Smart Innovation, Systems and Technologies book series (SIST, volume 44)

Cite this paper as:
Rai G.N., Gangadharan G.R. (2016) Architectural Characterization of Web Service Interaction Verification. In: Nagar A., Mohapatra D., Chaki N. (eds) Proceedings of 3rd International Conference on Advanced Computing, Networking and Informatics. Smart Innovation, Systems and Technologies, vol 44. Springer, New Delhi

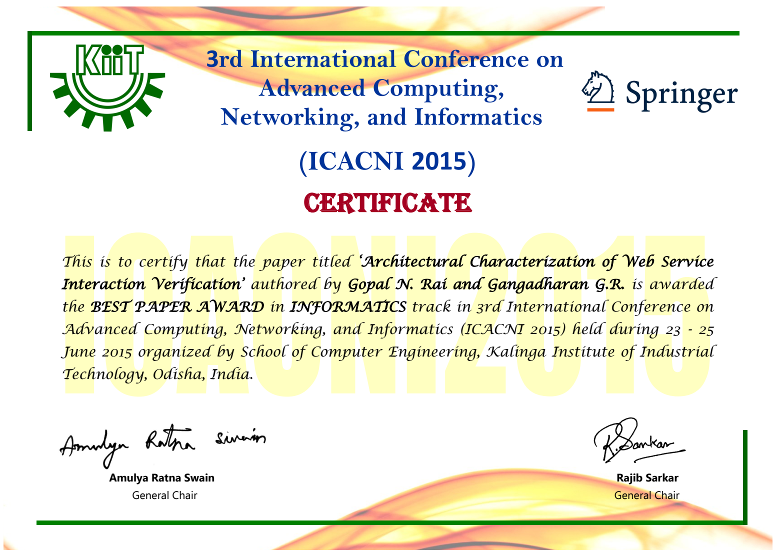
Abstract

Web service interaction utilizes disparate models as it still does not have its own model for verification process. Adaptation of a different model is not always beneficial as it may prune

Figure A.3: Publication [4]

Table A.3: Fact Sheet for Publication [4]

Title	Architectural Characterization of Web Service Interaction Verification
Authors	Gopal N. Rai and G. R. Gangadharan
Publication	In Proceedings of 3rd International Conference on Advanced Computing, Networking and Informatics: ICACNI 2015 pages 447-456
Online ISBN	978-81-322-2529-4
DOI	$10.1007/978-81-322-2529-4\_47$
Status	Published
Publisher	Springer India
Publication Type	Conference Proceedings



3rd International Conference on Advanced Computing, Networking, and Informatics

Springer

(ICACNI 2015)

CERTIFICATE

*This is to certify that the paper titled '**Architectural Characterization of Web Service Interaction Verification**' authored by **Gopal N. Rai and Gangadharan G.R.** is awarded the **BEST PAPER AWARD** in **INFORMATICS** track in 3rd International Conference on Advanced Computing, Networking, and Informatics (ICACNI 2015) held during 23 - 25 June 2015 organized by School of Computer Engineering, Kalinga Institute of Industrial Technology, Odisha, India.*

[illegible]

Amulya Ratna Swain
General Chair

[illegible]

Rajib Sarkar
General Chair

Figure A.4: Best Paper Award for the Paper Titled *Architectural Characterization of Web Service Interaction Verification*

Synopsis of

Modeling and Verification of Web Services Using Recursive Composition Algebra

Thesis submitted for the degree of

Doctor of Philosophy
in
Computer Science
by

Gopal Narayan Rai **Reg. No. 12MCPC05**

Under the supervision of

Dr. G. R. Gangadharan

Institute for Development and Research in Banking Technology
(Established by Reserve Bank of India)
Hyderabad-500057

School of Computer & Information Sciences
University of Hyderabad
Hyderabad-500046
Telangana, India

1 Introduction

Service-Oriented Computing (SOC) is a well-established computing paradigm developed over time and still passing through phases of technological refinements day by day. SOC utilizes software services (called as Web services) as fundamental elements for developing and deploying distributed software applications. Web services are self-contained and self-describing modular applications that can be published, located, and invoked across the web [2]. They communicate with each other through the exchange of messages based on the XML standards. The concept of composability among Web services is one of the most crucial reasons for the success and popularity of the services. Service composition is perceived as the federation of a service with other remote services, specifying the participating services, the invocation sequence of services and the methods for handling exceptions [2]. Web services are categorized into basic and composite [39]. A basic Web service is self-contained and independent whereas a composite Web service is dependent on other Web services and based on the requirements, forms composition out of available services. In the context of Web services, based on the structure, two types of composition are possible: linear composition and recursive composition [2, 17]. In linear composition, the constituent Web services are only basic Web services, whereas in the case of recursive composition, the constituent Web services could be basic as well as composite. The notion of recursive composition requires special attention [20] in a verification process as it is not easily tractable with classical modeling and verification schemes such as model checking and Petri net.

Designing and running dynamic Web service compositions are error-prone because a single service may have several dependencies with other services to perform their tasks correctly and a developer may not know the identity of those services that would fulfill the request. Analogous to other distributed systems based on asynchronous communication, it is difficult to anticipate how Web service compositions behave during execution and whether they conform to the functional requirements. Interaction among composite services through messages opens the space for concurrency related bugs. These concurrency bugs are difficult to find by testing, since they tend to be non-reproducible or not covered by test cases. Furthermore, the asynchronous nature of communications complicates the scenario and it becomes very difficult to analyze and debug. Existing approaches including model checking, Petri net, π calculus, artificial intelligence based algorithms, *Temporal Logic of Actions* (TLA), and case-based reasoning achieve the desired aspects of verification by modeling, planning, and verifying the Web services. However, these approaches have their own limitations. For instance, *Symbolic Model Verifier* (SMV) [9] is an input language for model checking and it does not support modeling subtleties regarding Web service interaction such as automatic discovery and dynamic availability of services [16]. Moreover, automatic composition and dynamic reconfiguration of Web services makes modeling and verification process more difficult.

Therefore, in this thesis, we propose: (i) a formal modeling technique, namely *Recursive Composition Algebra (RCA)* to capture the notion of recursive composition among Web services, and (ii) RCA based verification techniques for verifying interactive and structural properties of the services. Further, we provide implementation details and a cross-domain application of the proposed framework.

2 Problem Statement, Aim and Objectives

2.1 Problem Statement

Let $\mathcal{W}$ be a dynamic (the number of elements in the set may vary with time) set of Web services. Consider that at a time instance t, there are m number of services in the set $\mathcal{W}$ ($\mathcal{W} = \{w_1, \cdots, w_m\}, m \in \mathbb{N}$). Let $\mathcal{W}_B$ be a set of basic services and $\mathcal{W}_C$ be a set of composite services made out of the set $\mathcal{W}$ such that $|\mathcal{W}_B| + |\mathcal{W}_C| = m$. A Web service w_i consists of a set of input messages $w_i.I$ and a set of output messages $w_i.O$ and there is a pre-defined relation from input messages to output messages ($Rl \subseteq w_i.I \times w_i.O$). A Web service $w_i \in \mathcal{W}$ can interact with another Web service $w_j \in \mathcal{W}$ based on input output compatibility that provides a number of different ways to form a composition. This composition may trigger a sequence of actions and result in various interaction patterns (traces). Out of the all possible traces, some traces may be undesirable. Provided a set of available services, our goal is to verify structural and interactive properties by observing all possible traces while supporting automation in composition formation and dynamism in the set $\mathcal{W}$.

2.2 Aim

To provide formal modeling and verification techniques for verifying interactive and structural properties of Web services.

2.3 Objectives of the Thesis

1. To get insight into the Web service verification methods focused on composition verification, interaction verification, and compositional equivalence verification.
2. To understand the Web service verification requirements that needs to be addressed while developing a verification technique.
3. To develop a comprehensive modeling technique that can work as an interpretation model for composition, interaction, and compositional equivalence verification techniques.
4. To develop verification techniques for the Web service composition, interaction, and compositional equivalence based on the developed comprehensive modeling technique.

3 Literature Review

3.1 Web Service Composition Verification

Research works in the area of Web service composition verification spans across different approaches, such as: state transition model based [3], logic based [8], process algebra based [13], and Artificial Intelligence (AI) based [21]. The problem of automatic Web service composition generation is closely related to the problem of *Goal-Oriented Action Planning (GOAP)* in AI [5, 28, 34]. In literature, several AI planning based techniques for automatic composition are available: STRIPS-based [28], PDDL-based [19], HTN-based [44], etc. Recent AI planning based works such as [21] and [46] are better than previous proposals as they handle dynamic availability of services and domain-dependency of planning in more efficient way. Another line of work [1, 20] investigated into the recursive composition of Web services to find a better composition solution. Abrougui et al. [1] used recursive multi-agent systems to support dynamism in Web service composition. Jaiswal et al. [20] used a recursive composition based model to form a cost-effective composite service. However, verifying Web services composition automatically or with very less human intervention is still a challenging issue.

3.2 Web Service Interaction Verification

Nakajima [32] identified the faulty flow descriptions (written in *Web Services Flow Language*) by using software model-checking. Application of model checking based techniques for verification of Web service interaction is bit old [14, 15, 43], but still is in use because of its efficacy. Techniques presented in [14, 15, 43] were efficient, however, they did not deal with automation of verification process. Schlingloff et al. [37] and Zheng et al. [45] presented an integrated technique for modeling and automated verification of Web service composition. Their modeling was based on Petri net and for correctness they employed model checking technique with *Alternating Temporal Logic* (ATL). Rossi [35] proposed a model checking algorithm based on a logic-based technique for the verification of security and correctness properties of service interaction. Further, as an improvement over the previous ones, the recent model-checking based verification techniques [3, 12, 18, 38] support automation to a great extent. Although the existing solutions are promising, core techniques adopted in the solutions do not capture all the required characteristics of Web service interaction verification as they were natively proposed for different scenarios and applications.

3.3 Compositional Equivalence Verification between Web Service Composition Graphs

To the best of our knowledge, we do not find a work directly focused on the verification of compositional equivalence between Web Service Composition Graphs (WSCGs). However, our work is closely related with the context-dependent substitutability verification of Web services. Since substitutability is not a stand-alone expression, it is always studied with other accompanying terminologies such as compatibility [6, 30], behavioral equivalence [7, 25], and conformance [27]. The compatibility between two services ensures that they interact properly without any error. Klai et al. [22] used *Symbolic Observation Graph* (SOG) to determine compatibility between two services. Klai and Ochi [23] also used the SOG for checking compatibility of Web services behaviorally. Corrales et al. [10] accepts the input as two BPEL or two WSCL documents. Unlike to previously discussed compatibility-dependent notions, Pathak et al. [33] presented the concept of environment based substitutability notions: environment dependent and environment independent. Santhanam et al. [36] proposed a variation of the approach in [33]. However, they differ from [33] in their consideration of non-functional attributes for substitutability. Stahl et al. [40] proposed an environment-based solution for service substitutability. Kuang et al. [24] modeled a Web service and context using π-calculus. Further, they introduced a notion of behavioral equivalence by considering the context of a service. Runtime checking of service substitutability based on satisfiability of *Linear Temporal Logic* (LTL) specifications was proposed in [4]. Taher et al [41] proposed an interface-matching based solution for the problem of service substitutability. Liang et al. [26] also followed a similar approach of classifying Web services using their descriptions. Motahari-Nezhad et al. [31] advocated that checking of service equivalence should not be done merely on the basis of interface, its business protocol also must be considered. Another interface-based similarity measurement between services was studied in [42]. These kinds of verification could also be done by using the concept of subgraph matching. For instance, subgraph matching is used in [10, 29] for matchmaking of service behavior. Since the concept of subgraph isomorphism or subgraph bisimulation fall in the class of NP-Complete problems [11], researchers have been making effort to reduce the associated computational complexity by using the techniques such as *optimal error correcting subgraph isomorphism detection* [10]. However, there is still space for further reduction of the complexity.

4 Proposed Contents of the Thesis

This thesis consists of 9 chapters including an introductory chapter and a concluding chapter. The content of each of these chapters is summarized as follows:

4.1 Chapter 1: Introduction

Chapter 1 provides the motivation and the necessary background for the work reported in this thesis. The chapter investigates the primitive characteristics of Web service composition/interaction model that need to be well considered in the verification process. Also, it provides an overview of various verification approaches that were used to verify the Web services. This chapter concludes by providing the scope of the thesis.

4.2 Chapter 2: A Systematic Literature Review on Model Checking Based Web Service Verification

Chapter 2 conducts a Systematic Literature Review (SLR) on model checking of Web services. Model checking is a popular formal technique facilitating automatic verification of finite-state systems, and it has been applied for almost all of the Web service verification aspects, such as control-flow, data-flow, interaction, time requirements, quality of service, security requirements, etc. However, no systematic literature review focused on model checking of Web services is available. Motivated by this fact, in this chapter, we systematically review existing research works on model checking based Web service verification appeared during the period of 2002 – 2017. We review 122 referred research articles, and for each paper mentioned in this review, we identify the verification goal, target Web service technology, flavor of the employed model checking technique, tool, and other helping techniques, if any. Further, we highlight some of the issues, gaps, key challenges in this area and proposed some future directions.

4.3 Chapter 3: Recursive Composition Algebra (RCA)

In order to provide a rigorous and sound foundation for the formal reasoning about Web services, algebraic modeling is one of the important techniques used as is witnessed from the Web service literature. However, the algebraic modeling approach for Web services (Web service algebra) is still in its infancy. To further facilitate the algebraic modeling of Web services, in this chapter, we propose a composition algebra based on the notion of recursive composition. The proposed algebra is fully capable to verify the presence of behavioral equivalences and deadlock conditions in a Web service composition scenario. The main motivation for proposing a Web service composition algebra is to capture the recursive nature of composition which cannot be done using traditional approaches like model checking and Petri net.

4.4 Chapter 4: Web Service Composition Verification Using RCA

In this chapter, we present a formal model for reasoning and verifying dynamic Web service composition at design level. Our aim is to provide a methodology that takes a set of candidate Web services, their respective *Web Service Description Language* (WSDL) files, and their interaction specifications as input and provides the output whether an interaction specification is satisfied or not. Furthermore, if the interaction specification is not satisfied, then the counterexample is produced. The verification technique uses the concepts of *Web Services Set Partitioning* (WSSP) and trace. The WSSP based technique makes a set of candidate Web services being considered for a composition scenario into a number of subsets based on service invocation possibility and thereby generating a WSSP graph using RCA. As a part of verification technique, we use abstract modeling in the beginning that further leads to detailed modeling if required. This technique reduces the computation time and modeling complexity. The applicability of our proposition is validated by a travel agency case study.

4.5 Chapter 5: Web Service Interaction Verification Using RCA

In this chapter, we present a recursive composition based modeling and verification technique for Web service interaction. The application of recursive composition over a Web service with respect to the set of available Web services yields a *Recursive Composition Interaction Graph* (RCIG). In order to capture the requirement specifications of a Web service interaction scenario, we propose *Recursive Composition Specification Language* (RCSL) as a requirement specification language. Further, we employ the proposed RCIG as an interpretation model to interpret the semantics of a RCSL formula. Our verification technique is based on the generation and analysis of the exhaustive possible interaction patterns. Performance evaluation results, provided in this chapter, show that our proposition is implementable for the real world applications. The key advantages of the proposed approach are: (i) it captures primitive characteristics of Web service interaction patterns, such as, recursive dependency, sequential and parallel flow, etc., (ii) it does not require explicit system modeling as in model checking based approaches (NuSMV, SPIN, etc.), and (iii) it supports automatic composition and dynamic reconfiguration of services. The proposed mechanism is implemented and evaluated for the exhaustive possibilities in a travel agency case study.

4.6 Chapter 6: Compositional Equivalence Verification between Web Service Composition Graphs Using RCA

Given a composition request, the formation of possible *Web Service Composition Graphs* (WSCGs) depends on the set of available services. Since the availability of Web services is dynamic, at any time, a new service can join or an existing service can leave the set of available services. A change in the set may bring the structural change in a previously formed WSCG. However, this is not always the case that a structural change in the WSCG brings the semantic change. In this chapter, our aim is to verify the compositional equivalence between two WSCGs formed before and after the structural change caused by the change in the set of available services. Our proposed solution is based on an algebraic formalism, and by using the formalism, directed acyclic WSCGs are formed for a given composition request. Then, by using WSCGs, we propose the concept of *composition expression* and *canonical composition expression*. Based on the proposed concept of canonical composition expression, we verify compositional equivalence between two WSCGs. The advantage of our approach is that it reduces the equivalence verification to the subsumption checking between two algebraic expressions instead of directly using the WSCGs and solve a subgraph matching problem. The proposed mechanism is implemented and evaluated for the exhaustive possibilities in a travel agency case study with respect to a given composition request.

4.7 Chapter 7: An Integrated Tool for Web Service Verification

In this chapter, we present a tool for modeling and verification of Web services. The tool is implemented in Java language. Experiments are carried out in lab conditions, that is, Web service repositories are deployed on the systems connected to a Local Area Network. There are three verification modules in the tool as per the requirement: composition verification, interaction verification and compositional equivalence verification. The core technique used for modeling of Web services is recursive composition algebra (RCA). Application of RCA on a service or message generates the recursive composition graph (RCG) that works as a building block in the formation of interpretation model for the verification process. In order to specify the verification requirements, recursive composition specification langauge (RCSL) is used.

4.8 Chapter 8: Application of RCA in Network Security

Detection of multistage Cyber attacks in an enterprise network is of growing interest to the security community. Existing threat modeling techniques, such as attack graphs and attack trees, do not facilitate reasoning about the causal relationship between network vulnerabilities. Consequently, the area of multistage attack needs more exploration. In this chapter, we propose a multistage attack modeling technique (namely RCA_{MA}) based on the recursive composition algebra. For a given vulnerable network configuration, the algebra generates recursive composition graph (RCG) which depicts all possible multistage attack scenarios. The prime advantages of the RCG is that it is free from cycles, therefore, does not require computation intensive cycle detection algorithms. Further, canonical sets obtained from the RCG classifies network vulnerabilities into five classes: (i) isolated, (ii) strict igniter (entry point), (iii) strict terminator (dead end) (iv) overlapping, and (v) mutually exclusive. These classes (logical inferences) provide better insight into the logical correlation among existing vulnerabilities in a given network and hence in prioritizing vulnerability remediation activities accordingly. The efficacy and applicability of our proposition is validated by the means of a case study.

4.9 Chapter 9: Conclusions and Future Directions

Chapter 9 summarizes the contributions of the thesis and outlines the future directions. The following research directions are suggested for future:

- Although our proposed interaction verification approach is completely able to achieve its objectives, it still has two limitations: partially solved state explosion problem and non-consideration of *Quality of Service* properties. Therefore, addressing these limitations is one of our future works.

- Our proposed compositional equivalence verification technique could be improved by providing the support for: (1) writing requirement specification and verification of that, (2) generic kind of communication model rather than being focused only on the Web services. The participating entities for this communication model could be software components or intelligent agents (as in artificial intelligence) and the model could be able to capture the multi-party asynchronous communication.

5 List of Articles Published During the Candidature

5.1 Papers in Conference Proceedings

1. Gopal N. Rai, G. R. Gangadharan, and Vineet Padmanabhan. "Algebraic Modeling and Verification of Web Service Composition." In *Proceedings of the 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015)*, London, UK, pp. 675-679. 2015. [Indexing: DBLP, Scopus]

2. Gopal N. Rai, and G. R. Gangadharan. "Set Partition and Trace Based Verification of Web Service Composition." In *Proceedings of the 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015)*, London, UK, pp. 278-285. 2015. [Indexing: DBLP, Scopus]

3. Gopal N. Rai, and G. R. Gangadharan. "Architectural Characterization of Web Service Interaction Verification." In *Proceedings of the 3rd International Conference on Advanced Computing, Networking and Informatics (ICACNI)*, Bhubaneswar, India, pp. 447-456, 2016. [Indexing: Scopus] [**BEST PAPER AWARD**]

5.2 Papers Under Review

1. Gopal N. Rai, G. R. Gangadharan, Vineet Padmanabhan, and Rajkumar Buyya. "Web Service Interaction Modeling and Verification Using Recursive Composition Algebra." Submitted to: *Journal of Network and Computer Applications*. Elsevier.

2. Gopal N. Rai and G. R. Gangadharan. "Verifying Compositional Equivalence between Web Service Composition Graphs." Submitted to: *Concurrency and Computation: Practice and Experience*. Wiley.

3. Gopal N. Rai, Ghanshyam S. Bopche, G. R. Gangadharan, and Babu M. Mehtre. "Modeling and Analyzing Multistage Attacks Using Recursive Composition Algebra." Submitted to: *Computers & Security*. Elsevier.

REFERENCES

[1] Anissa Abrougui, Annabelle Mercier, Michel Occello, and Marc-Philippe Huget. Recursive multi-agent system for dynamic and adaptative web services composition. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, MEDES '09, pages 44:295–44:299. ACM, 2009.

[2] Gustavo Alonso, Fabio Casati, Harumi A. Kuno, and Vijay Machiraju. *Web Services - Concepts, Architectures and Applications*. Springer, 2004.

[3] Jamal Bentahar, Hamdi Yahyaoui, Melissa Kova, and Zakaria Maamar. Symbolic model checking composite web services using operational and control behaviors. *Expert Systems with Applications*, 40(2):508–522, 2013.

[4] Marcello M Bersani, Luca Cavallaro, Achille Frigeri, Matteo Pradella, and Matteo Rossi. SMT-based verification of LTL specification with integer constraints and its application to runtime checking of service substitutability. In *Proceedings of the 8th International Conference on Software Engineering and Formal Methods (SEFM)*, pages 244–254. IEEE, 2010.

[5] Piergiorgio Bertoli, Marco Pistore, and Paolo Traverso. Automated composition of web services via planning in asynchronous domains. *Artificial Intelligence*, 174(3):316–361, 2010.

[6] Lucas Bordeaux, Gwen Salaün, Daniela Berardi, and Massimo Mecella. When are two web services compatible? In *Technologies for E-Services*, pages 15–28. Springer, 2004.

[7] Andrea Bracciali, Antonio Brogi, and Carlos Canal. A formal approach to component adaptation. *Journal of Systems and Software*, 74(1):45–54, 2005.

[8] Federico Chesani, Paola Mello, Marco Montali, and Paolo Torroni. Verification of choreographies during execution using the reactive event calculus. In *Proceedings of the Web Services and Formal Methods*, pages 55–72. Springer, 2009.

[9] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *Proceedings of the Computer Aided Verification*, pages 359–364. Springer, 2002.

[10] Juan Carlos Corrales, Daniela Grigori, Mokrane Bouzeghoub, and Javier Ernesto Burbano. Bematch: a platform for matchmaking service behavior models. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, pages 695–699. ACM, 2008.

[11] Agostino Dovier and Carla Piazza. The subgraph bisimulation problem. *IEEE Trans. Knowl. Data Eng.*, 15(4):1055–1056, 2003.

[12] Warda El Kholy, Jamal Bentahar, Mohamed El Menshawy, Hongyang Qu, and Rachida Dssouli. Modeling and verifying choreographed multi-agent-based web service compositions regulated by commitment protocols. *Expert Systems with Applications*, 41(16):7478–7494, 2014.

[13] Andrea Ferrara. Web services: a process algebra approach. In *Proceedings of the 2nd International Conference on Service oriented computing*, pages 242–251. ACM, 2004.

[14] Howard Foster, Sebastian Uchitel, Jeff Magee, and Jeff Kramer. Model-based verification of web service compositions. In *Proceedings of the 18th IEEE International Conference on Automated Software Engineering*, pages 152–161. IEEE, 2003.

[15] Xiang Fu, Tevfik Bultan, and Jianwen Su. Analysis of interacting BPEL web services. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 621–630. ACM, 2004.

[16] Holger Giese and Basil Becker. *Modeling and verifying dynamic evolving service-oriented architectures*, volume 75. Universitätsverlag Potsdam, 2013.

[17] Carlos Granell, Michael Gould, Roy Grønmo, and David Skogan. Improving reuse of web service compositions. In *E-Commerce and Web Technologies*, pages 358–367. Springer, 2005.

[18] H. Groefsema, N. van Beest, and M. Aiello. A formal model for compliance verification of service compositions. *IEEE Transactions on Services Computing*, 10.1109/TSC.2016.2579621, 2016.

[19] Ourania Hatzi, Dimitris Vrakas, Mara Nikolaidou, Nick Bassiliades, Dimosthenis Anagnostopoulos, and Ioannis Vlahavas. An integrated approach to automated semantic web service composition through planning. *IEEE Transactions on Services Computing*, 5(3):319–332, 2012.

[20] Vimmi Jaiswal, Amit Sharma, and Akshat Verma. Recomp: Qos-aware recursive service composition at minimum cost. In *Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011)*, pages 225–232. IEEE, 2011.

[21] Eirini Kaldeli, Alexander Lazovik, and Marco Aiello. Domain-independent planning for services in uncertain and dynamic environments. *Artificial Intelligence*, 236:30–64, 2016.

[22] Kais Klai and Hanen Ochi. Checking compatibility of web services behaviorally. In *Proceedings of the 5th International Conference on Fundamentals of Software Engineering, FSEN*, pages 267–282, 2013.

[23] Kais Klai, Hanen Ochi, and Samir Tata. Formal abstraction and compatibility checking of web services. In *Proceedings of the 20th International Conference on Web Services (ICWS)*, pages 163–170. IEEE, 2013.

[24] Li Kuang, Yuxin Mao, and Yingjie Xia. A decidable formal analysis of context-specific behavioral equivalence for web services. *Advanced Science Letters*, 4(4-5):1687–1694, 2011.

[25] Li Kuang, Yingjie Xia, Shuiguang Deng, and Jian Wu. Analyzing behavioral substitution of web services based on pi-calculus. In *Proceedings of the IEEE International Conference on Web Services (ICWS)*, pages 441–448. IEEE, 2010.

[26] Qianhui Liang, Bu-Sung Lee, and Patrick CK Hung. A rule-based approach for availability of service by automated service substitution. *Software: Practice and Experience*, 44(1):47–76, 2014.

[27] Fangfang Liu, Yuliang Shi, Liang Zhang, Lili Lin, and Baile Shi. Analysis of web services composition and substitution via CCS. In *Data Engineering Issues in E-Commerce and Services*, pages 236–245. Springer, 2006.

[28] Shiyong Lu, Arthur Bernstein, and Philip Lewis. Automatic workflow verification and generation. *Theoretical Computer Science*, 353(1):71–92, 2006.

[29] Shang-Pin Ma and Jonathan Lee. A graph-based approach to web service matchmaking. In *Proceedings of the 19th Asia-Pacific Software Engineering Conference*, pages 796–801. IEEE, 2012.

[30] Axel Martens. On compatibility of web services. *Petri Net Newsletter*, 65(12-20):100, 2003.

[31] Hamid Reza Motahari Nezhad, Guang Yuan Xu, and Boualem Benatallah. Protocol-aware matching of web service interfaces for adapter development. In *Proceedings of the 19th international conference on World wide web*, pages 731–740. ACM, 2010.

[32] Shin Nakajima. Verification of web service flows with model-checking techniques. In *Proceedings of the First International Symposium on Cyber Worlds*, pages 378–385. IEEE, 2002.

[33] Jyotishman Pathak, Samik Basu, and Vasant Honavar. On context-specific substitutability of web services. In *Proceedings of the IEEE International Conference on Web Services (ICWS)*, pages 192–199. IEEE, 2007.

[34] Joachim Peer. *Description and automated processing of web services*. PhD thesis, University of St. Gallen, 2006.

[35] Sabina Rossi. Model checking adaptive multilevel service compositions. In *Formal Aspects of Component Software - 7th International Workshop, FACS*, pages 106–124, 2010.

[36] Ganesh Ram Santhanam, Samik Basu, and Vasant Honavar. Web service substitution based on preferences over non-functional attributes. In *Proceedings of the International Conference on Services Computing. SCC'09.*, pages 210–217. IEEE, 2009.

[37] Holger Schlingloff, Axel Martens, and Karsten Schmidt. Modeling and model checking web services. *Electronic Notes in Theoretical Computer Science*, 126:3–26, 2005.

[38] Quan Z. Sheng, Zakaria Maamar, Lina Yao, Claudia Szabo, and Scott Bourne. Behavior modeling and automated verification of web services. *Information Sciences*, 258:416–433, 2014.

[39] Quan Z Sheng, Xiaoqiang Qiao, Athanasios V Vasilakos, Claudia Szabo, Scott Bourne, and Xiaofei Xu. Web services composition: A decade's overview. *Information Sciences*, 280:218–238, 2014.

[40] Christian Stahl and Karsten Wolf. Deciding service composition and substitutability using extended operating guidelines. *Data & Knowledge Engineering*, 68(9):819–833, 2009.

[41] Yehia Taher, Djamal Benslimane, Marie-Christine Fauvet, and Zakaria Maamar. Towards an approach for web services substitution. In *Proceedings of the 10th International Database Engineering and Applications Symposium, IDEAS'06*, pages 166–173. IEEE, 2006.

[42] Okba Tibermacine, Chouki Tibermacine, and Cherif Foudil. A practical approach to the measurement of similarity between wsdl-based web services. In *Proceedings of the 6ème Conférence francophone sur les Architectures Logicielles, CAL*, pages 3–18, 2012.

[43] Christopher Walton. Model checking multi-agent web services. In *Proceedings of the AAAI Symposium of Semantic Web Services*, 2004.

[44] Dan Wu, Bijan Parsia, Evren Sirin, James Hendler, and Dana Nau. Automating daml-s web services composition using shop2. In *Proceedings of the International Semantic Web Conference*, pages 195–210. Springer, 2003.

[45] Yongyan Zheng, Jiong Zhou, and P. Krause. A model checking based test case generation framework for web services. In *Proceedings of the 4th International Conference on Information Technology (ITNG '07)*, pages 715–722. IEEE, 2007.

[46] Guobing Zou, Yanglan Gan, Yixin Chen, and Bofeng Zhang. Dynamic composition of web services using efficient planners in large-scale service repository. *Knowledge-Based Systems*, 62:98–112, 2014.