PhD Dissertation

International Doctorate School in Information and Communication Technologies

DIT - University of Trento

FORMAL ANALYSIS OF WEB SERVICE COMPOSITIONS

Raman Kazhamiakin

Advisor:

Prof. Marco Pistore

Università degli Studi di Trento

March 2007
To my family
First of all I would like to thank my advisor Marco Pistore for his priceless support and guidance throughout these years. Without his ideas, comments, and friendly assistance this dissertation would not have been ever possible. Marco helped me to understand how to carry out the research, exemplifying the energy, curiosity and eagerness to achieve the goals. I learned a lot from him, and I very grateful for this invaluable experience and the opportunity to work together.

I would also thank my colleagues and collaborators, especially Marco Roveri, Paritosh Pandya, Luca Santuari, and the great people of the Astro project, for making the work really amazing, and for creating a very friendly, encouraging and fruitful environment.

Special thanks go to my reviewers, Davide Sangiorgi, Fabio Casati, Paritosh Pandya, and Paolo Traverso, who have made a great effort to read the thesis and provided useful feedback.

I wish to say a lot of good words to my friends and fellow students for their support and concern. I would particularly express my thanks to Andrei Tamlín for being true friend all this time.

With my deepest gratitude I dedicate this work to my family: to my parent for everything they have given to me, for their belief and encouragement, and to my wife Tania for giving me inspiration, understanding, and love.
Abstract

One of the key ideas of Web service technology is the ability to create service compositions by combining pre-existing services using standard languages for the definition of the composition behavior. These compositions often span across the enterprise boundaries, involving various stakeholders with their own requirements and goals into the design process. The ability to detect requirements violations and resolve conflicts in the specifications of the composition behavior has been an important issue in the service-oriented design. This makes the automation of the correctness analysis a challenging research topic for the development of critical and reliable service-based applications.

The formal analysis of the Web service compositions has to tackle several problems specific to this domain. First, the service communications are essentially asynchronous and rely on complex and heterogeneous message management systems. Second, the service specifications extensively use complex data structures and operations, often in a non-deterministic and partially defined manner. Third, the correctness of the service-based processes often relies on quantitative properties, especially on time aspects of the execution. These issues, however, are not systematically addressed by the existing approaches, and require specific methods and solutions.

In this dissertation we present a formal framework for the analysis of Web service compositions that supports asynchronous interactions and complex data and time flow modelling. The framework relies on a formal
model, where (i) services are defined as state transition systems equipped with data and time characteristics; (ii) services are composed using a parametric communication model, which allows for capturing a wide range of messaging systems; (iii) temporal logics are exploited for the specification of behavioral requirements, taking into account also quantitative time aspects. Based on this model we introduce new analysis approaches that deal with issues specific to the Web service domain.

We develop a technique to associate with a Web service composition the most adequate communication model, i.e., the one that is sufficient to capture all the behaviors of the composition, while being as efficient as possible in the analysis, and allows for extracting minimal constraints on the underlying implementation.

We propose an analysis approach that takes into account the data flow among the component process. The approach exploits abstraction techniques for modeling only relevant data flow aspects. We show that building the right abstraction corresponds to iterative extraction of certain assumptions on the partially defined or even unknown data manipulations performed by the component services.

We also present a timed analysis approach that allows for the verification of complex time properties and for the computation of duration bounds where these properties are satisfied, using the duration calculus formalism.

These techniques are implemented and integrated in a toolkit that allows for model checking Web service compositions, and are evaluated on a set of case studies.

Keywords
Web service composition, verification, model checking
Contents

1 Introduction .................................................. 1
   1.1 Contributions ........................................... 6
       1.1.1 Analysis Framework ............................... 6
       1.1.2 Addressing Specific Problems .................. 9
       1.1.3 Prototype Tool and Experimental Results ....... 13
   1.2 Thesis Outline ........................................... 15

2 State of the Art .................................................. 17
   2.1 Service-Oriented Engineering ......................... 17
       2.1.1 Web Services .................................. 18
       2.1.2 Service-oriented Architecture .................. 20
       2.1.3 Web Service Composition ....................... 21
       2.1.4 Analysis of Web Service Compositions .......... 24
   2.2 Correctness Analysis of Software Systems ............ 27
       2.2.1 Formal Specification ............................. 28
       2.2.2 Formal Analysis ................................ 29
   2.3 Related Work ............................................ 31
       2.3.1 Approaches Based on Process Algebra .......... 31
       2.3.2 Approaches Based on Petri Nets ................ 33
       2.3.3 Approaches Based on State Machines .......... 34
       2.3.4 Logic-based Approaches ......................... 36
   2.4 Discussion ............................................... 38
### 3 Formal Model

3.1 Conceptual Model of the Composition .................................. 41
3.2 Formal Model of Web Service .............................................. 44
   3.2.1 Data Model ....................................................... 44
   3.2.2 Time Model ...................................................... 46
   3.2.3 STS Formalism .................................................. 47
3.3 Formal Model of Web Service Composition ............................. 49
   3.3.1 Modelling Composition Structure ................................... 49
   3.3.2 Modelling Message Interactions ................................... 49
   3.3.3 Global Transition System ......................................... 51
   3.3.4 Composition Behavior ............................................. 53
   3.3.5 Infiniteness of the GTS Model ..................................... 54
3.4 Discussion ................................................................. 55

### 4 Representing Web Service Specifications ................................. 57
4.1 Modelling BPEL Processes .................................................. 57
   4.1.1 Overview of BPEL .................................................. 58
   4.1.2 From BPEL to STS .................................................. 61
4.2 Modelling Composition Requirements ..................................... 72
   4.2.1 Linear-time Temporal Logic ....................................... 73
   4.2.2 Interpretation of the Property Categories ......................... 75
4.3 Discussion ................................................................. 77

### 5 Analysis of Communication Models ........................................ 79
5.1 Case Study: Virtual Travel Agency ....................................... 80
   5.1.1 Example 1: Tickets Reservation Scenario ......................... 80
   5.1.2 Example 2: Reservation with Cancellation ....................... 82
   5.1.3 Example 3: Extended Cancellation ............................... 84
5.2 Problem Formalization ..................................................... 85
5.3 Hierarchy of Models ....................................................... 86
List of Tables

4.1 Mapping basic BPEL activities to STS .......................... 63
4.2 Mapping structured BPEL activities to STS ...................... 66
4.3 Mapping flow BPEL activity to STS ............................. 68
4.4 Mapping timed BPEL activities to STS .......................... 69
4.5 Mapping fault and compensation BPEL activities to STS . 70

7.1 Mapping duration annotations to STS ............................ 144

8.1 Partial order reduction ........................................... 167
8.2 Adequacy analysis results ................................. 170
8.3 Scalability: NuSMV results ................................. 172
8.4 Scalability: SPIN results .................................... 173
8.5 Loan Approval: Experimental results .......................... 180
8.6 Timed VTA: Verification results ............................. 182
8.7 Timed VTA: Computation results ............................ 183
8.8 WMO: Experimental results ................................. 187
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Prototype analysis tool</td>
<td>14</td>
</tr>
<tr>
<td>2.1</td>
<td>Web Services stack</td>
<td>20</td>
</tr>
<tr>
<td>2.2</td>
<td>Service-Oriented Architecture</td>
<td>21</td>
</tr>
<tr>
<td>2.3</td>
<td>Generic Web service composition</td>
<td>24</td>
</tr>
<tr>
<td>4.1</td>
<td>Flight Reservation BPEL process</td>
<td>60</td>
</tr>
<tr>
<td>5.1</td>
<td>Virtual Travel Agency composition: Example 1</td>
<td>81</td>
</tr>
<tr>
<td>5.2</td>
<td>Virtual Travel Agency composition: Example 2</td>
<td>83</td>
</tr>
<tr>
<td>5.3</td>
<td>Virtual Travel Agency composition: Example 3</td>
<td>84</td>
</tr>
<tr>
<td>5.4</td>
<td>Bounded composition with unbounded skeleton</td>
<td>104</td>
</tr>
<tr>
<td>5.5</td>
<td>Inadequate model is adequate for the skeleton</td>
<td>105</td>
</tr>
<tr>
<td>6.1</td>
<td>Loan Approval Example</td>
<td>113</td>
</tr>
<tr>
<td>7.1</td>
<td>Waste management application</td>
<td>141</td>
</tr>
<tr>
<td>7.2</td>
<td>Excerpt of the STS with time constructs</td>
<td>145</td>
</tr>
<tr>
<td>7.3</td>
<td>Example of complex timed requirement</td>
<td>148</td>
</tr>
<tr>
<td>7.4</td>
<td>$\text{AnyOnce}(st)$ transition system</td>
<td>158</td>
</tr>
<tr>
<td>8.1</td>
<td>WS-VERIFY: functional architecture</td>
<td>163</td>
</tr>
<tr>
<td>8.2</td>
<td>Data analysis: performance scalability</td>
<td>176</td>
</tr>
<tr>
<td>8.3</td>
<td>VTA: timed cancellation</td>
<td>181</td>
</tr>
<tr>
<td>8.4</td>
<td>Timed analysis: performance scalability</td>
<td>184</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

“Nothing is stronger than its weakest part.” (C. Kingsley, 1856)

In the past decades, the evolution of software systems observes continuous growth of the complexity levels and the maturation of the networks and protocols. The cross-application integration and interoperability, the flexibility of implementation and reconfiguration became essential requirements for the modern technology solutions and architectural styles. These requirements encourage methods and approaches that support effective system design and provision and allow for high degree of modularization and component reuse. The evolution of the architectural styles continuously supported this trend, having resulted in service-oriented approaches.

The Service-Oriented Architecture, together with Web service technology, concentrates on the definition of a software system as a complex distributed business application that involves a composition of independent software agents. These agents provide their functionality as the services available on the Web, and collaborate together with an aim of achieving a set of business goals and providing certain business capabilities. The set of Web service standards allow for the formal definition of the different functional and non-functional aspects of the composition.

The complexity of the composition model, its structure, and interactions between the involved services require careful assessment of the possible sys-
tem behavior. Moreover, the composite applications often span across the enterprise boundaries, thus involving a wide range of different stakeholders into the system modelling process. These stakeholders often come into play with their own requirements and policies that may be inconsistent or even contradictory in certain scenarios. The quality of the designed system, the correctness of its behavior, and hence the success of procurement, directly depend on the ability to ensure that the composition specification satisfies the desired business requirements of all the interested parties.

While these issues are also applicable to the design of traditional distributed systems, SOA and Web service technology exhibit some important differences. First, the use of standards for service descriptions provides an ability to focus on business-level requirements and functionalities and to abstract from the low-level technology- and implementation-related issues. This separation of business models from technical details simplifies the system representation, allows for involving even non-technical experts into the system design process, and therefore simplifies the problem of modelling and analyzing service compositions and composition requirements. On the contrary, the tight coupling of the business requirements and the technical aspects in early distributed systems was difficult to manage and often led to a long and error-prone design process.

Second, Web service technology aims at creating highly autonomous and reusable components and compositions. The services are designed with the goal to be used by different clients and in different settings, while satisfying internal requirements and policies. The more correct scenarios and use cases are allowed for the service, the higher value is achieved by the service provider. The possibility to model and analyze different settings and scenarios is vital in order to understand the relation between the designed system and the business requirements prior its deployment.

In these circumstances, the approach to the design and development
of the SOA-based distributed business processes should incorporate the formal modelling and analysis techniques that enable the correctness assessment of the system behavior [72]. In our vision, such an approach should be able to solve the following tasks:

- Formal modelling of the behavioral composition specifications;
- Formal representation of various business requirements and policies;
- Validation of the composition specification;
- Verification of requirements against the composition specification.

Moving towards an analysis approach that supports the design and development of Web service composition, it is important to take into account the features that are peculiar to SOA and Web service technology. These features not only require specific analysis techniques, but also demand specific approaches and methods specific ways to model composition requirements, to search for the problems, and to encapsulate the analysis results into the system development process. We consider the problems related to modelling and analyzing service communications, data management, and timed properties as one of the most important for the composition analysis.

**Web service interactions.** Unlike the traditional distributed systems, the Web services are loosely coupled and autonomous. They interact in an asynchronous way: different services send and receive messages concurrently and independently from each other. The messages go through different layers of software, and hence through multiple queues, before they are actually consumed by the component implementing the service. This functionality is provided by a certain message delivery middleware that the service platform supports.
This, however, considerably complicates the behavioral analysis of the Web service composition. Indeed, due to the presence of complex queueing mechanisms, concurrent message overpasses, the behavior of the composition and the results of the analysis strictly depend on the formal model adopted for the representation of the message delivery middleware. On the other hand, this model is implementation-dependent, may vary in several ways (i.e., queue ordering, number and structure), and, moreover, is abstracted away from the composition specification.

In these settings, the ability to reason on service interactions in a generic way, to understand the effect that various message delivery models have on the composite behavior, and to infer properties of the middleware that are important for the given composition becomes an essential requirement for the Web service composition analysis.

**Data flow complexity.** One of the strongest capabilities of Web services is the use of XML for representing, exchanging, and transforming business data in a uniform and flexible way. The Web service specification standards allow for the definition of a data flow among the component services. Clearly, the data flow is as important for the static analysis of Web service compositions as the control flow, but in most of the existing approaches to the formal analysis of service compositions (see, for instance, [53, 102]) the data is abstracted away, thus potentially invalidating the analysis results.

The necessity to manage data, however, brings the following problems to the composition analysis. First, the data domains are often infinite (e.g., user-defined types in XML documents), and the semantics of data manipulating operations is complex (e.g., the XPath expressions used in composition specifications). This restricts the applicability of traditional formal techniques that rely on simple and finite system representations.

Second, due to the distributed nature of the Web service applications
and high level of abstraction, the internal details of the service implementations may be hidden from the analysts complicating the proof of correctness. As a result, specific techniques and approaches are required in order to manage incompleteness and infiniteness of the data.

**Time-related issues.** In the behavioral analysis of Web service compositions we require not only the satisfaction of qualitative requirements (e.g., deadlock freeness of the interaction protocols), but also of quantitative properties, such as time, performance, and resource consumption. Time-related properties are particularly relevant in these settings. Indeed, in many scenarios we expect that the composition satisfies some global timed constraints, and these constraints can be satisfied only if all the services participating to the composition commit to respect their own local timed constraints. Consider an e-government scenario, where the distributed application requires the composition of information systems and functionalities provided by different departments or organizations. The composite service can comply with the timed commitments with respect to the national regulations only if they are consistent with the time required by all participating actors to carry out their part of the process.

In the analysis of such properties it is important not only to check whether a certain requirement is satisfied, but also to determine extremal time bounds where the property is guaranteed to be satisfied. For example, in e-government scenario it is important not only to demonstrate the ability to complete the authorization procedure, but also to find a maximal/minimal duration of such a procedure. Design of the corresponding algorithms and models that take time issues into account is a must in order to analyze a wide class of Web service composition scenarios.
1.1 Contributions

In this dissertation we develop an approach to formal representation and analysis of Web service compositions. The goal of this approach is to support the composition design process, providing a way to detect possible problems in the specification before the composition is being actually deployed.

In order to achieve this goal, we introduce a unified framework that integrates formal specification and formal reasoning capabilities. Special analysis techniques and methodologies are developed within the framework so as to tackle the challenges specific to the Web service technology and Service-Oriented Architecture. A prototype tool implements our framework and the presented methods and algorithms.

1.1.1 Analysis Framework

The presented framework defines a formal model for the representation of Web service compositions, and a set of formal methods and algorithms for the analysis of their behavior.

**Formal model.** We model a service composition as a closed system consisting of a finite set of service models that interact with each other through a formal model of the message delivery medium, namely communication model. The communication model defines a set of message queues (of possibly infinite capacity), their structures and ordering constraints. This model is used to represent the complexity of the service interactions that take place during the composition execution. Each service model is represented as a State Transition System, that is used to represent the dynamic aspects of the service behavior. In this formalism, the service can be in one of its states, and evolves to a new state performing some ac-
tions. These actions may represent the interactions of the service (i.e.,
the emission and reception of a message) or certain internal computations
(i.e., modification of internal variables, conditional choices, event handling,
etc.). The behavior of the composite system is, therefore, defined as a set
of sequences where common states of the component services and their
internal/communication actions are interleaved.

The presented formal model of the Web service composition allows for
the definition of the following aspects:

- **control flow** of the composition that defines the evolution of the com-
  posite system that performs actions and moves from a state to another;

- **data flow** of the composition that defines the business data of the
  services, its modification and exchange during interactions;

- **time flow** of the composition that represents the timed properties of
  the model, such as duration of operations and time-related events
  (e.g., timeouts and deadlines).

**Representation of composition specification.** In practice, a Web service
composition is specified using a set of standard languages. These languages
allow for the description of various aspects of the model, such as interface
definition, behavioral model, non-functional properties. In order to apply
formal analysis techniques, a mapping between the standard specification
and the formal model should be defined. Here we define a translation pro-
cedure that allows us to represent the Web service specification in terms
of the State Transition System formalism. The specification is defined as a
WSDL document [137] representing the service interface, and a BPEL [5]
document that represents the service behavior. The choice of BPEL as a
language for the definition of service behavior is dictated by the following
reasons. First, BPEL is a widely accepted standard being developed by
1.1. CONTRIBUTIONS

OASIS [105], and is widely supported by industry implementations\(^1\). Second, BPEL is one of the most expressive languages for the description of the composition behavior [144]. It provides a highly expressive notations for the representation of, e.g., transactional behavior, event handling, data management, etc. We remark, that the formalism presented in this work may be easily adapted for the representation of other composition languages, such as WS-CDL [142], WSCI [138], BPML [109].

**Formal verification.** Formal analysis techniques presented in this thesis are based on *model checking* [35], an analysis approach that, given a formal representation of the analyzed system and a formal definition of the behavioral requirements, performs an exhaustive exploration of all the behaviors of the system. The intensive research in the field of model checking entailed a set of highly efficient algorithms and techniques and led to a successful application of this analysis approach for the verification of software. Model checking approach allows not only for checking whether the property is satisfied or not, but also generates a counterexample which represent the execution of the system that violates the required property. This approach is extremely useful in the domain of Web service compositions, where the testing approaches fail due to the distributed nature of the composition and inability to freely test the target application.

**Methodological support for the analysis process.** Building upon the verification techniques, we present a set of analysis methodologies that guide the analysis tasks and continuously support the composition development. These methodologies allow the analyst to operate at the level of composition specification and business requirements, hiding the details on transformation, low-level representation, and actual reasoning activities. This

\(^1\)At the moment of writing this thesis about 30 industrial implementations exist.
applies also to the analysis results, which may be moved upwards in analogous way.

First, we propose an approach to infer the properties relevant for modelling service interactions. The approach allows us to detect possible message losses and unbounded message queue growth, to determine features of the messaging mechanisms that may affect the correctness of the system, and to find a model of communications that enables the most efficient but complete composition verification. Second, we present an iterative analysis and refinement process able to cope with the infiniteness and incompleteness of data. The process allows not only for the verification of the composition specification, but also for modelling and extracting data requirements on the service internals that are necessary for the correctness of the composition. Finally, the timed analysis approach is focused on modelling of high-level time-related properties and requirements.

### 1.1.2 Addressing Specific Problems

In order to address specific problems of the Web service composition domain, we introduce in our framework a set of novel analysis methods and techniques. In particular, we present an approach to deal with complex asynchronous message exchanges; provide a rigorous way to specify and analyze various data and time properties of the composition behavior.

**Analysis of communication models.** The formal model we use in our framework is based on a parametric definition of the communication infrastructures. More precisely, it is possible to define different communication models by changing the number of queues existing among the component processes, the sets of messages associated with the various queues, their ordering constraints and bounds. Modifying these parameters, e.g., increasing the number of queues, and hence allowing more and more asynchrony in
the evolution of the system, we define a hierarchy of communication models that are able to model a potentially infinite range of the composition scenarios [81, 87]. The most restrictive model, with only one shared queue of capacity 1, is shown to be equivalent to a model, where the partners synchronize on shared actions, namely synchronous model, which is widely used for service composition modelling. The most liberal model, instead, which has dedicated queues for each type of message, can describe virtually all the examples of BPEL compositions we found in the literature and in practical usage.

We also present an algorithm for reasoning on interaction mechanisms in BPEL compositions. The algorithm is able to identify the simplest communication model in the hierarchy that is adequate for a specific set of BPEL processes, i.e. the model that allows for the representation of all the executions foreseen under an arbitrary messaging mechanism. In this way, the approach enables not only the most complete representation of the composition, but also allows for abstracting from the implementation details of the underlying middleware. Moreover, the identification of the simplest adequate model allows us also to determine the set of requirements that the queueing system should satisfy in order to guarantee the correctness of the system, or, in other words, to identify an appropriate middleware implementation.

The outcome of the algorithm is then used to build the corresponding composition, that can be used for further analysis, such as model checking of behavioral requirements against the composition model. The experimental results show that the performance of the verification decreases when the complexity of the communication model increases, and that the possibility to select automatically the right model is very useful to improve the verification performance.
Data-flow analysis. Facing the data-challenges, we present an analysis approach that allows us to represent and manage arbitrary data structures, and overcome the problem of the information incompleteness [84]. The approach is based on abstraction techniques [33, 63] that take into account only those properties that are relevant for the verification, and discard the aspects that are irrelevant. The incompleteness of knowledge about service implementations is managed through an iterative analysis process, where the verification of the control and data requirements is interleaved with the elicitation of the assumptions on the missed knowledge that ensure these properties. At the end, this will produce a refined model, where the specification is enriched with a set of assumptions and constraints that are crucial for the system correctness, and where all the expected properties have been formally verified. While these assumptions and constraints are discovered and collected during the static, design-time verification of the compositions, they may be further exploited for the dynamic, run-time analysis of the compositions using monitoring techniques.

More precisely, the analysis approach presented here addresses the following issues:

- Provide automated analysis techniques based on data abstractions for the specification and verification of different behavioral properties. For these purposes we define two models of abstractions that “bound” the behavior of the system and answer the verification query with certain precision in presence of complex or even unknown data functions and operations. These precision may be improved by refining the abstractions and adding new and new facts (or abstract predicates) to the composition model.

- Manage the incompleteness in the composition specification allowing the definition of assumptions on the hidden properties of the sys-
1.1. CONTRIBUTIONS

tems. These assumptions provide a way to clarify the semantics of the data manipulation operations and functions, and to represent the expectation of the system designer about the (potentially unknown) implementation. At run-time these properties may be monitored by the interested party in order to validate the correctness of the composition.

- Support an iterative refinement process by combining the abstraction-based verification with the requirements extraction. In this process, when the satisfiability of a property can not be determined, the model is refined by adding new predicates or new assumptions on the internals of the service operations. The process is repeated until the property is guaranteed to be satisfied or a trace that demonstrates the violation of the property is found.

Analysis of timed properties. The results described here extend the traditional quantitative analysis approaches with the ability to represent, verify and even compute timed characteristics of the Web service composition [80, 79]. On the one hand, the presented formalism allows for specifying time flow in the behavior of the composition. This is achieved through the special timed semantics of the service execution, in which the system evolves and time potentially passes. On the other hand, we allow for modelling timed requirements on the composition at the level of business processes. We extend the capabilities provided by the Web services standards (e.g., modelling timeouts in BPEL) with an ability to model duration of basic operations, decisions, or even complex subprocesses. We also exploit the duration calculus logic [28] for the representation of complex timed requirements in the domain, such as timing of events, satisfaction of properties on subintervals, intervals sequencing, etc. We want to stress the fact that the time properties we want to model and analyze are those that are
critical from the point of the business logic, i.e., the time to carry out a task or to make a decision, and the constraints on these times that guarantee a successful execution of the distributed business processes. The “technical” times, which are required, for instance, for the communications and management of messages, are orders of magnitude smaller (seconds if not milliseconds) and can be neglected in such scenarios.

Based on these modelling concepts, we provide techniques for the formal analysis of timed properties of the Web service composition. In particular, the techniques allow for model checking the compositions, and incorporate the analysis of both traditional temporal properties and timed requirements. For these purposes we adapt Quantified Discrete-time Duration Calculus (QDDC, [113]) and provide a finite-state representation of the underlying system, which is then used as input for the model checker.

Finally, we introduce a decision procedure that can be used to compute extremal durations of intervals satisfying required (timed and non-timed) properties. We use the QDDC logic to model these properties and present symbolic algorithms based on results of [115, 23] to perform the computation in our model.

1.1.3 Prototype Tool and Experimental Results

**WS-VERIFY toolkit.** The analysis techniques and approaches presented here are implemented and incorporated into a prototype tool WS-VERIFY. The toolkit is integrated with the Active Webflow Designer\(^2\), a visual framework for the design and development of BPEL processes, extending it with formal analysis capabilities. The architecture of the toolkit is represented in Fig. 1.1.

The specification of the Web service composition is translated into the State Transition System format. Currently, the composition specification

\(^2\)http://www.active-endpoints.com
is accepted in the form of a set of BPEL and WSDL files, however, other standards may be thought of, provided that the corresponding translator is defined.

After the translation, the analysis of communication models is applied, and the analysis results are provided to the composition designer. This step may be skipped, and some predefined communication model is then used. Given the data-related properties and constraints as input, the tool may build an abstraction of the composition specification. In this phase the variables of infinite domains, functions and operations are abstracted away following certain model of abstraction. As before this step may be omitted by the user, provided, however, that all data domains are finite. If the timed analysis should be applied, the timed properties of the model are extracted and transformed into the specific representation accepted by the model checker, otherwise the timed properties (if any) are ignored and this step is skipped.

Finally, the obtained specification is augmented with the behavioral properties to be verified, such as temporal logic queries, complex timed
requirements, time bounds computation requirements, etc. The resulting model checker specification is used as input to the model checker, which performs the verification and provides the results. Currently the tool supports two state of the art model checkers, namely NuSMV [30] and SPIN.

Experimental evaluation. In order to provide a definitive account of the presented analysis approach and incorporated methods and techniques, we evaluated our implementation on a set of case studies emerged from industrial applications and found in the literature.

The purpose of the experimental evaluations is twofold. First, we used the tool to instantiate the analysis approaches and methodologies presented in this work. The results of the experiments demonstrated the viability of our analysis techniques, and permitted detecting and refining some non-trivial problems in the composition models. Second, we evaluated efficiency and scalability of the presented reasoning techniques. Based on the obtained results, we were able to find and improve the tool implementation, applying various optimization techniques. Final experimental results presented here, demonstrate good performance of the analysis even for complex composition scenarios.

1.2 Thesis Outline

The rest of the dissertation is organized as follows. In Chapter 2 we give an overview of the state of the art in the area of Web service composition specification and analysis. This overview starts from the background information that introduces modern standards and approaches for the description of Web services and service compositions. The chapter proceeds with the discussion of the current trends and technologies being applied for the formal analysis of software systems in general, and concludes with
the review of the corresponding works and research lines in the specific area of the Web services and service-oriented technology. Chapter 3 provides a guided description of the formal model, upon which the analysis techniques and approaches presented here are based. In Chapter 4 we define the mapping of the Web service specifications (in particular, BPEL and WSDL models) into the formalism, and introduce the language for expressing behavioral requirements. The following chapters present the formal techniques and approaches used for the analysis of Web service compositions. In particular, Chapter 5 presents the methodology for the analysis of the communication mechanisms in Web service compositions. Chapter 6 describes the abstraction-based approach for the analysis of data and data flows in the compositions, and Chapter 7 presents an approach for the modelling of timed properties of Web services, their verification, and computation algorithms. Chapter 8 presents the implementation of the WS-VERIFY toolkit in details, as well as the results of certain experiments related to the presented analysis techniques. Concluding remarks, and a future work is discussed in Chapter 9.
Chapter 2

State of the Art

In this chapter we give an overview of the service-oriented computing paradigm, Web services technology and Web service composition design. We discuss the specific problems of the composition development and their influence on the correctness analysis of composite applications. We also present existing approaches and techniques for the verification of software systems, and review the related works in the filed of the analysis of Web service compositions and services-oriented architectures.

2.1 Service-Oriented Engineering

Increasing complexity of the information systems, necessity to cope with the growing rate of change in the market landscape, necessity for the IT organizations to effectively manage and adapt their business to respond quickly to these changes have been driving the evolution of the distributed computing. Addressing these challenges, distributed computing passed through the client-server model [46], multi-tier architecture [45], RPC-based standards (e.g., CORBA [108]), having finally reached the Service-Oriented Architecture. Today IT environment aims at achieving efficiency and flexibility of operationalization across and beyond enterprise boundaries, thus enabling their seamless interoperability [116]. The main chal-
2.1. SERVICE-ORIENTED ENGINEERING

The challenge is to simplify the integration of business functionalities, regardless of possible heterogeneity of implementing platforms, protocols, and devices.

Service-Oriented Architecture exploit services as the main constructs in order to deliver low-cost, flexible, and scalable distributed business applications. The intensive utilization of standards for the description, discovery, and invocation of Web services, on the one hand, allows one to hit the problem of integration, and on the other hand, to abstract from the low-level technology details thus enabling the integration and management at the level of business functionality.

Due to intrinsic complexity and breadth, the problem of engineering service-oriented solutions has to face the challenges that origin from different disciplines, such as networking, knowledge representation, software engineering, artificial intelligence [116]. It is also important to take into account the features that are specific to SOA in order to provide adequate and efficient products. Moreover, the ignorance of these features and direct re-application of existing solutions potentially lead to a failure in the design and procurement of the service-oriented business applications. In the following we describe the Web service technology and the SOA approach, and discuss their specific potential pitfalls, related to the problem of the correctness analysis of the service-based processes and applications.

2.1.1 Web Services

A Web service is merely a virtual software component accessible via multiple protocols. W3C defines Web services as follows [139]:

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in
a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

The goal of the technology is to provide a natural support for platform-independent component invocation through a simple, well-defined and commonly understood interface, designed with the expectation of reuse.

The key aspect of this technology is the use of standardized protocols and languages, designed to improve integration between the cooperating parties. These standards cover virtually all aspects of the development and provisioning, such as representation and exchange of business data, description of functional and non-functional characteristics of a service, aggregation, coordination, and management of a group of services.

Figure 2.1 represents a stack of standards constituting Web service architecture. In particular, Simple Object Access Protocol (SOAP, [64]) provides a framework for packaging and exchanging XML messages that can be carried by a variety of network protocols, such as HTTP, SMTP, FTP, RMI/IIOP, or a proprietary messaging protocol. Web Service Description Language (WSDL, [137]) allows for the definition of the service functionality, that is the set of service operations, their input/output parameters (message definitions), and specific data types being used in message exchanges. These data types are represented using XML Schema ([141]), a rich notation for the description of complex hierarchical data structures.

Whenever it is necessary to describe how the internal state of the service changes on performing different methods, or to specify a interaction protocol that one should follow in order to correctly use service functionality, the service model may be augmented with some behavioral description. This abstract behavior may be specified, e.g., in Business Process Execution Language (BPEL, [5]), a notation for the representation and execution of service-based workflows.
2.1. SERVICE-ORIENTED ENGINEERING

<table>
<thead>
<tr>
<th>Composition</th>
<th>Management</th>
<th>Security</th>
<th>QoS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negotiation, SLA, policies</td>
<td>WS-Policy, WS-Policy Assertions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Choreography</td>
<td>WS-CDL, WSCI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transactions</td>
<td>WS-CAF, WS-Transactions, WS-Coordination</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orchestration</td>
<td>WS-BPEL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Discovery and publishing</td>
<td>UDDI, WSIL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functional description</td>
<td>WSDL, abstract BPEL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data description</td>
<td>XML, XML Schema</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Messaging</td>
<td>SOAP, WS-Reliability, WS-Addressing</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.1: Web Services stack

2.1.2 Service-oriented Architecture

Service-Oriented Architecture (SOA) supports a programming model that allows services to be published, discovered and invoked by each other in a platform-, protocol-, and language-independent manner [88]. Integration and infrastructure management are the key elements for the architecture that uses services as basic constructs for easy composition of distributed business applications.

Conceptual model of SOA is represented in Fig. 2.2. A typical instantiation of this model is described as follows. A service is implemented, described, and published by a service provider in a UDDI (Universal Description, Discovery, and Integration [106]) registry. A service requestor queries the registry in order to find a service that implements some required functionality. Once a suitable service is found, the requestor obtains the service description, binds to the provider, and invokes service operations.

This generic procedure may vary in different ways [140]. The description of the service may cover additional aspects, such as security and quality of
service characteristics, semantic annotations, usage policies; the discovery may be manual or automated, registry-based or peer-to-peer; the binding may be dynamic or static, occasional or permanent; the service selection may use a wide range of functional and non-functional criteria, involve a certain negotiation process between parties. In order to achieve this diversity, basic steps and descriptions may be insufficient. A web service standards stack is therefore extended with additional layers and notations, that cover higher level aspects, such as service aggregation, coordination, management, quality of service, security, etc.

With this generic and flexible model, SOA naturally addresses the problem of cross-enterprise application integration. Distributed and independent Web services may be easily composed together into a new business application, regardless specific implementation platforms and technologies. This application may be further published as a service, creating a complex collaboration network, referred to as service composition [89].

2.1.3 Web Service Composition

Web service compositions are traditionally described using the terms orchestration and choreography [119]. The former focuses on the representa-
tion of the protocol of a particular participant to a composition, while in
the latter the composition is considered from the global perspective, defin-
ing a collaboration of actors that interact in order to accomplish a common
goal. Accordingly, different design approaches are assumed for these views.
The orchestration foresees the bottom-up development, where the compo-
sition is constructed from a set of local specifications. It is often referred
to an executable process that interacts with other services accomplishing
the goals of the orchestrator. On the contrary, the choreography assumes
a top-down design, where the global specification of the composed system
is being developed. This global model may be seen as a blueprint of the
composite behavior that the real applications should conform to.

Composition languages. There is a wide range of industrial standardization
efforts towards providing specification languages for the Web service com-
positions. Among them BPEL (Business Process Execution Language, [5])
and CDL (Web Service Choreography Description Language, [142]) are the
most used orchestration and choreography languages respectively. Regard-
less a particular notation, the languages address the following problems:

- The ability to represent various behavioral properties of the compo-
sition. This includes the necessity to model service interactions (i.e.,
  message exchanges); control flow constraints (i.e., sequencing, condi-
tional branching, iterative execution of application activities); data
  flow constraints (i.e., exchange, modification, evaluation of data and
data expressions).

- The ability to handle different application execution modes. That is,
apart from modelling normal flow of the execution, the language is
  aimed at representing exceptional and transactional behavior, provid-
ing facilities to coordinate, recover, and compensate abnormal behav-
ior of the system. The problem of transactional support is particularly challenging in the service composition domain, since the classical transactional mechanisms fail in the distributed, loosely-coupled context [129]; there is a need to deal with so-called Long-Running Transactions (LRT), where business processes may last for days, weeks or even months (e.g., supply-chain domains, e-government applications). This requires the supporting specification to speak explicitly about time flow, time constraints, timeouts.

- The ability to represent the compositions at a high level of abstraction. Indeed, the composition specification language is more likely to be a tool for business programming rather than for a traditional way of application coding. This requires an ability for modularization, abstraction, coarse granularity of activities. As a consequence, the ability to express non-determinism becomes essential in order to conceal certain implementation details, model a freedom of choice, or to abstract from a certain application logic.

Conceptual model of the service composition. Web service composition may be represented in the following generic way (Fig. 2.3). The composition describes a collaboration of a set of interacting participants. Each participant is described with a set of specifications that define its interface, data formats, usage policies, quality of service parameters, and other properties.

The behavior of the composition may be represented as a unique global model (e.g., WS-CDL choreography specification), or composed from the local behavior specifications of the participants (e.g., BPEL protocols).

The participants exchange messages using a complex communication medium implemented by underlying middleware systems and responsible for message transmission, delivery, and queuing. These middleware systems represent various technologies and platforms, run various message
2.1. SERVICE-ORIENTED ENGINEERING

Figure 2.3: Generic Web service composition

transport protocols, such as HTTP, SMTP, RMI-IIOP, etc. A concrete implementation is abstracted away from the composition specification and may vary both for the composition and for the participants instances.

2.1.4 Analysis of Web Service Compositions

Over last few years the problem of the formal analysis of Web service compositions has been widely recognized as an important and challenging problem [72, 122, 52]. The necessity to integrate independent applications and to negotiate between independent stakeholders, complexity of the composite model, potential incompleteness of both the service and the platform descriptions, failure to apply traditional correctness assessment techniques (such as software testing) require a rigorous and accurate analysis approach to ensure the correctness of the resulting composition.

In general, the analysis approach aims at checking correctness of the composition specifications with respect to a certain set of functional and non-functional requirements, ranging from high-level strategic business goals and needs to low-level technical constraints and policies. Apart from vertical classification, these requirements may be classified horizontally: global requirements express the business properties of the composition
model as a whole, while local requirements express the expectations, offers, and obligations of a participant to a composition [140]. This makes the problem of understanding, modelling, unrevealing, and analyzing of Web service requirements one of the cornerstone problems for the design of SOA-based applications [91, 37, 110], and requires integration of business requirements models in the business process life cycle [86, 85, 3].

We are interested in particular in the analysis of behavioral requirements, i.e. the properties that the composition should satisfy in order to achieve its functional goal. These requirements include (but indeed are not limited to) freeness of deadlocks in the composition execution, conformance to another composition model, correct manipulation and transformation of data, obeying rules and constraints on ordering between interactions, correct termination, error and transaction failure recovery. Such requirements may be defined automatically (e.g., deadlock freeness), extracted from the specification (e.g., correct termination), or defined explicitly using some formal or semiformal notation, such as Message Sequence Charts specifications [74], temporal logic [48], etc.

**Formal analysis process.** The analysis procedure for the formal assessment of the composition specification consists of several tasks.

First, the analyzed specification is given a formal, i.e. precise and unambiguous, interpretation. A wide range of formal models exists that allow for such a representation. Indeed, the way the system is executed (i.e., the behavior of the system) should be formalized as well.

Second, the requirements should be given an interpretation in terms of the formalism applied to the composition specification. If for instance, the behavior of the composition is formalized as a set of execution sequences, then the correct termination is interpreted as a necessity to reach certain state on every execution in the set.
2.1. SERVICE-ORIENTED ENGINEERING

Third, an analysis algorithm is applied in order to check the formal representation of the requirement against the formal representation of the composition. The algorithms depend on the formal model applied for the representation of the specifications, on the kind of requirements to be analyzed. Together with the yes/no answer, the analysis may produce some additional information, such as a counterexample trace that demonstrates an execution of the system, where the requirement is violated.

The choice of the formalisms, analysis techniques, and their application depends on the way the composition is being developed.

**Top-down design and analysis.** In the top-down design approach, abstract specification of the composition is the outcome. This specification is later populated with the real service implementations. In this case, it is important to verify that the composition model is able to satisfy behavioral requirements. Whenever the specification is too abstract, the analysis may fail since some important properties about the future implementations are not known. It is therefore needed to make certain assumptions about unknown details that would guarantee the correctness of the implementation. These assumptions become requirements to the implementation of the abstract model.

When the specification is being instantiated with the real services, the composition analysis is applied as follows. First, it is necessary to check that the implementation conforms to the abstract specification (conformance testing [66]). Second, it is necessary to verify that the implementations satisfy previously assumed properties. Related to this is the problem of the service replaceability analysis [104, 20], which aims at checking whether a service may be replaced by another service while still achieving the desired correctness of the composition.
Bottom-up design and analysis. In the bottom-up development approach the composition is being constructed from the specifications of existing services, resulting in an executable distributed application. The analysis applied for such a strategy includes the composability/compatibility analysis [24, 20], the verification of the business goals and requirements against the resulting composition, etc.

In the later phases of the service composition life-cycle, such as the composition maintenance, the correctness analysis of the composition behavior is used to ensure the traceability of business requirements, that is to determine how the changes of the composition affects business requirements and goals.

2.2 Correctness Analysis of Software Systems

During the whole history of the software/hardware system design, formal methods have been intensively used for proving correctness of the systems. Started from assertional proofs of Goldstine, von Neumann, and Turing, and passed through theories of computations of McCarthy, flowchart methods of Floyd, Naur, and, finally, results of Hoare, research on reasoning about program correctness matured as a wide area of computer science [78]. This area produced a lot of correctness analysis approaches, theories, languages, and algorithms, including theorem proving [27], automata-theoretic methods [128], temporal logic [48], model checking [35], etc. Due to advances in this fields (see, e.g., [97, 33, 38]) over past decades, the formal correctness analysis has been successfully applied in the area of hardware verification [47, 43, 123], and gains increasing spread in the area of software analysis [98, 117, 34]. Recent successful results in applying formal analysis for real systems (e.g., verification of correctness of driver code by Microsoft [9]) advocate the applicability of these techniques, and
motivated the Hoare’s claim about the automated verification of program correctness as a *Grand Challenge for Computing Research* [70]. Below we present a brief classification of the formal analysis approaches according to the models used to represent and reason over the systems, and to the formal methods applied.

### 2.2.1 Formal Specification

In the area of formal methods, the way the system is being modelled and described is always driven by the problem being addressed by the formal approach. Whenever the main concern is precise system representation, the preference is given to the form of the model, while the analysis is treated as secondary. The formal methods like Z [125] or VDM [77] focus on formal modelling of sequential systems, and provide reach mathematical notations and constructs for representing functional and non-functional aspects of the system. Instead, the primary focus of the Process Algebra-based formalisms [100, 69, 73, 13] and Petri nets [120] is the representation of concurrency and mobility in distributed software systems. Various extensions were also proposed to deal with data, time, probability, etc. The ability to deal with concurrency and flexible formal model led to a wide use of Petri Nets and Process Algebra as formal models for distributed systems and business processes, and recently for Web services and service compositions.

In other approaches, however, the main focus is on the analysis automation, while the formalism used for the system representation is less important. This is the case, for instance for the approaches based on automata, Statecharts, or finite state machine formalisms [61, 68, 26, 4]. These models usually provide a simplified and restricted formal notation, making difficult a representation and traceability of high-level aspects (e.g., transactionality, mobility, complex data structures), but allow one to focus
on the efficient implementation of the analysis techniques. Several models exist that allow for the formalization of certain important aspects. For example, timed automata [4] allow representation and reasoning on timed properties, Promela language [71] allows for (bounded) data structures and communication middleware (queue systems).

Another way to describe systems is provided by logic-based approaches. In these approaches the system features, like, e.g., operations, structure, behavior, are represented as formula in some logic. For example, temporal logic [48] allows for the definition of partial ordering of events in the system behavior, constraints on the system executions, etc. Their real-time extensions, duration calculi or interval logics [28] allow for modelling behavioral properties explicitly referring to time. Logic-based modelling allows for partial, and incremental representation of the “system-to-be”, and, therefore is particularly suitable for the early phases of the system development [59]. These partial models representing required properties of the systems may be further exploited in the later phases of the system design, where the “system-as-is” is to be verified to satisfy them.

2.2.2 Formal Analysis

The existing formal analysis approaches differ in several ways [34]. First, they may be distinguished by the machinery used for the verification. There are two well-established methods, namely model checking and theorem proving.

Model checking [35] consists of building a finite representation of the analysed system and of its exhaustive exploration in order to guarantee that the desired property holds. This method is completely automatic, and, moreover, can provide a scenario that demonstrates the violation of the property. It can be used for the analysis of behavioral requirements, like temporal logic properties (temporal model checking), or conformance
between different models (simulation-based methods, refinement). The main challenges in model checking deal with the state explosion problem (addressed by symbolic model checking [97], partial order reduction [62], data abstraction [63], etc.), and with the analysis of certain classes of infinite state space systems.

In the theorem proving approach [27] the model and the property are represented as a system of axioms and inference rules. The analysis, therefore, consists of finding a formal proof of the property from this system. Opposite to model checking approach, theorem proving is often directly applicable for the analysis of infinite systems, but usually is not fully automatic, and requires continuous interaction with the analysts.

Second, the analysis methods and algorithms depend on a problem domain and a class of properties being analyzed. Specific methods were defined in order to deal with the processes interacting through infinite queues [19, 2], systems operating infinite data domains [63], dynamic systems [31], timed systems [4], etc. Indeed the domain of Web service compositions is one of the ultimate problem domains, as it positions the notions of interactions, data complexity, and time as a first-level entities in the correctness analysis. These kinds of systems are intrinsically infinite and can not be directly analyzed by traditional model checking. In this situation hybrid approaches that incorporate model checking with theorem proving represent the most prominent analysis strategy. Such approaches incorporate the full automation provided by model checking with the power of description and reasoning over infinite systems.

One of the most notable examples of hybrid approaches is the abstraction refinement [63, 32]. In abstraction-based verification a potentially infinite model of the system is replaced with a simple model that is being checked for a certain property. Under certain rules, the correctness of the abstraction implies the correctness of the real system. When the correct-
ness can not be ensured, the abstraction is automatically refined, using a certain system of axioms and inference rules.

These advances in the area of formal analysis research over the last decades led to a widespread adoption of automated verification methods both in hardware and software design. The success in their application to the real-size problem is achieved by the combination of different methods and algorithms, including symbolic software model checking, counterexample-guided abstraction refinement, compositional reasoning, etc (see, e.g., [9, 32, 29]).

2.3 Related Work

The problem of formal modelling and analysis of Web services and service compositions has recently become a focus of a wide number of research works. A part of these works mainly concerns the formalization of Web service composition specifications. This aspect is particularly important since many specification languages and standards lack formal semantics. Another research line is devoted to the development of formal analysis approaches, that would assist the design of Web service compositions.

In this section we overview various related works presented in the literature. We discuss the approaches and the results obtained with respect to the goals and problems we presented in the introduction to this thesis, namely the ability to represent and analyze Web service compositions, and to take into account the specific features of the problem domain. Further comparisons may be found in [99, 111].

2.3.1 Approaches Based on Process Algebra

Formalization. Various existing and new process algebras are used to formalize a generic service compositions (see, e.g., [50, 90, 25, 20, 52]), and to address certain specific issues, such as transactions and compensations
[21, 16], error handling [94], timed aspects [60, 93], mobility [18], correlations and session typing [136, 25, 18]. In this way some composition languages, like BPEL and WS-CDL, were provided with formal semantics.

An important feature of the Process Algebra approach is the ability to extend the model in order to capture a particular concept, such as transactionality or session typing. This allows for the native support of this concept and for deriving specific analysis methods. Such an extension also provides a two-way mapping between the designed system and the analysis results.

While some of the approaches take into account data [50] or timed behavior [60], the problems related to the asynchronous nature of service interactions is not addressed. The interactions are simply modelled as a rendezvous communications (the partners synchronize on shared actions), thus considerably restricting the class of the systems that may be correctly represented and analyzed.

**Analysis.** The process algebra models of Web service compositions may be analyzed by a variety of tools, such as CWB [36] for CCS, LTSA [92] for FSP, CADP [49] for LOTOS. The tools allow for different kinds of analysis. First, the above tools support verification of deadlocks, liveness and safety properties, CTL* model checking enables the analysis of different behavioral requirements. Second, the tools allow for behavioral equivalence checking based on various simulation/bisimulation relations. Equivalence checking opens up a possibility to perform service replaceability analysis, conformance checking, analysis of high-level aspects, related for instance to transactions or fault handling.

The presented approaches, however, rely on traditional analysis techniques, and usually omit the difficulties related to the asynchronous interactions, data complexity, analysis of timed properties. Asynchronous inter-
actions are usually replaced with synchronous; data-related constructs are usually abstracted away, or handled in a restrictive way, e.g., by bounding value domains. In [60] timed properties are analyzed by the translation to timed automata formalism [4] followed by the verification in UPPAAL timed model checker [10]. In [93], instead, it is shown how to represent timed constructs using non-timed ones under certain reasonable constraints.

### 2.3.2 Approaches Based on Petri Nets

**Formalization.** Petri Nets are widely used for the representation and reasoning over workflows and business processes [131]. With the standardisation of BPEL, Petri nets were exploited to model services and compositions (see, e.g., [111, 148, 126]). For this purpose both ordinary Petri nets and various dialects are used (workflow nets [135], Colored Petri nets [147], service/resource nets [126], hierarchical Petri nets [146]).

In the Petri nets formalism there is no semantic distinction between the interactions and internal service actions. In the discussed approaches service composition is, therefore, represented as a single Petri net, complicating the representation and analysis of message passing and interaction-related problems. A comparison between the process algebra and Petri nets as formalisms for web service compositions is investigated in [132].

**Analysis.** As well as process algebraic approaches, Petri net-based approaches are supported by the tools and allow for the definition of various analysis problems, such as temporal model checking, process simulation, etc. Additionally, in [135] the authors introduce the mapping from BPEL to workflow nets, and discuss several kinds of analysis of standalone BPEL process, namely detection of unreachable activities, detection of concurrent consumption of the same message, analysis of message types that may be
consumed in the rest of the execution.

The simplifications applied in case of process algebraic approaches, hold also for Petri Net-based approaches. The concepts of asynchronous communications, data and time flow are abstracted away or addressed in a very restricted form (data values enumerations in CP-Nets [147], timed semantics and time semantics interpretation using linear temporal rules in Service/Resource Nets [126]).

2.3.3 Approaches Based on State Machines

Formalization. A wide range of works on the Web service composition analysis deal with a formal model that can be referred to as state machines or automata. This is a rather low-level model that describes a service (as well as a composition) with a set of states or program locations. The behavior of the service is given with a set of transitions that specify how the program location is changed. The transition are labelled in order to distinguish the communication actions and internal actions. Notably, this model is used as a low-level semantic model for more complex formalisms, like Process Algebra.

The state machine model is also used in [12, 8, 145] in order to represent Web services. There only the interaction actions are represented, and the internal computations are abstracted away. In the composition the services synchronize on shared actions. This model may be given in terms of deterministic finite state automata. Indeed, such simplifications is applicable to a very restricted class of compositions.

This basic model may be extended in several ways. In [101, 56, 14] a service is equipped with typed variables, and the transitions are enriched with the enabling conditions (i.e., expressions over variables) and assignments (modification of variables) in order to represent stateful behavior of the service and data flow. The complexity of XML type system and
data operations is addressed in [57], where the authors provide a formal language for representing and mapping XML Schema. These models allow also for representation of internal operations, and for non-determinism of the resulting behavior model.

With respect to the problem of asynchronous communication the works of [56, 58] are particularly relevant. In that works the composition is represented as a set of service automata that exchange messages through a certain communication medium. This medium is given as a set of unbounded FIFO queues associated to the processes. This middleware structure is, however, fixed, which is not applicable in many realistic scenarios and is violated by several middleware implementations.

Similar to data, the model is extended with notion of time [67, 60]. In this case, the model defines also the passing of time when the system remains in the state or performs a transition.

**Analysis.** The research works that exploit state machines as a representation of Web services and compositions advocate temporal model checking as a primary analysis method. The behavioral requirements are represented in terms of temporal logics, and the verification is performed using a model checker, such as SPIN [56, 57, 101, 124, 85], NuSMV [121, 86, 6] or TLC [76].

Simulation checking is used in [8] to perform conformance test, that is to verify that the implementation of a composition participant conforms to the agreed composition protocol.

Apart from model checking, several research works present analysis techniques that target specific features of Web services. Synchronizability analysis presented in [56, 58] allows one to detect whether the asynchronous communication may be replaced with the synchronous one without loss of correctness. This approach allows for complete verification of a subclass of
systems with unbounded queues.

In [57] a verification approach that takes into account the complexity of the XML structures and operations is presented. These constructs are first translated to an intermediate formal language MSL, and then to Promela, the input language of the SPIN model checker, provided that the data structures and data types are finite.

In [42, 41] the verification of timed properties in Web service compositions is presented. The composition specification is translated to a network of timed automata, and verified using UPPAAL model checker.

2.3.4 Logic-based Approaches

**Formalization.** Various logic-based formalisms are widely exploited for modelling web services and service compositions in the area of Semantic Web Services [96]. The use of logic theories, in particular first-order theories, is dictated by the necessity of integration of various ontology models that are logic theories themselves. Such a formalisation allows for natural representation of the state of the world, and effect of the service execution on it. These properties are vital for the problem of finding an appropriate service, i.e., service discovery problem, which is one of the main focuses in the Semantic Web Services research. The set of first-order theories in terms of PSL is used in [65] for the formalization and axiomatization of DAML-S [96]. The logic-based formalisation of DAML-S is also presented in [103, 95] using situation calculi and logic programming language Golog. A specific logic system CTR-S, which is based on first-order logic, and a corresponding proof theory for reasoning on contracts for Semantic Web Services is presented in [39].

The power of logic theories is also exploited in [40] in order to model and analyze data-intensive Web services. In that approach the automata-like representation of service workflow is equipped with an abstract data base
that represents the state of the service. The evolution of the database is
given by a set of update rules represented as first-order logic sentences.
The analysis properties are represented in first-order temporal logic. The
analysis algorithms, however, are not considered and are left as a future
work.

Opposite to these approaches, the work of [15] defines a lightweight
logic-based framework, for representation of various constraints for Web
services. The focus in the work is made on the simplicity of constraints
and efficiency of the analysis. In this way the framework allows for par-
tial description of services, using simple propositional and temporal con-
straints on service calls. The approach that interleaves business processes
models with partial description in terms of temporal logic properties and
constraints is represented in [121, 86].

Logic-based approaches usually concentrate on the formalisation and
axiomatization of service descriptions, and do not consider the problems
related to the asynchronous interactions, or time flow.

**Analysis.** A variety of reasoning techniques are exploited in these ap-
proaches for the analysis purposes. In [103] the situation calculus model for
Semantic Web services is translated to Petri Net model, and various proper-
ties are verified using reachability analysis. The requirements specification
for Web services represented as a set of temporal logic properties [121, 86]
are also analysed using model checking techniques. The verification is ac-
complished by translating each temporal property to a non-deterministic
automaton, and then verifying a product in the NuSMV model checker.

The Web service interface framework of [15] presents a set of specially
designed algorithms for reasoning on partial service models and composi-
tions. In particular, the approach allows for checking compatibility between
services, replaceability, and refinement.
2.4 DISCUSSION

In a set of approaches a proof-based reasoning is adopted for the analysis of a service or a composition. The use of first-order theories, however, introduces additional complexity for the composition analysis and requires serious restrictions on the model. In [44] an algorithm for verification of WSFL workflow preconditions is presented. The approach is based on program logic, and the reasoning may be fully automatized under strong restrictions on data manipulation. In [39] the problem of behavioral contracting in multipart collaboration is addressed. A proof theory allows one to infer that the model of collaboration (contract) satisfies certain properties (constraints). The theory, however, requires the contract model to be acyclic and satisfy “unique-event” property, that is every event should occur only once on the contract execution. The restrictions are applied in [40], where a rule-based first-order theory is exploited for the modelling and analysis of data-intensive Web services. Strong restrictions on both the model and the analysis properties are required in order to make the reasoning feasible, if at all decidable.

2.4 Discussion

One of the key benefits provided by the Service-Oriented Architecture is that it allows for simple and flexible integration of pre-existing services into new distributed business applications, referred to as service compositions. Finding methods and approaches to assist the composition design process and to ensure composition correctness has been addressed by a wide range of research efforts.

Most of the presented approaches, however, suffer from the fact that many important factors and characteristics that are specific to SOA and Web service compositions are omitted or oversimplified. Among the most important are the asynchronous communications that rely upon different
protocols and communication systems; complexity of data structures and
data management in the composition; partial representation and knowledge
about the service functionality; strong relation between timed properties of
the models and transactional aspects of the composition behavior. Without
considering these aspects the formalisation and analysis efforts become very
restricted and immediately fail to address a wide range of realistic problems
and problem types.

In our research work we were primarily motivated by the necessity to
provide an analysis framework that is capable to manage the issues specific
to SOA and Web services. Like in many other related works, the goal of the
framework was to enable the analysis of behavioral requirements against
the composition models specified in a standard notation. However, we
extend the conventional formalisms and approaches in order to address the
specific issues of the problem domain. Indeed, such an extension entailed
the development of novel analysis methods and algorithms that are vital
for accomplishing this task.

**Formal model.** We used the state transition system as a model for the rep-
resentation of Web services for the following reasons. First, this model is
rather simple and may be used as an intermediate representation between
the verification and a higher-level specification, such as service description
in WSDL/BPEL, or a process algebraic formalization. Second, it is close
to the formalisms used in many state of the art analysis techniques, sim-
plifying their application and extension. We further extended this model
with the capabilities to represent infinite data types, arbitrary functions,
non-deterministic behavior, and timed properties hence enabling important
features of the service description.

In order to deal with asynchronous communications, we defined a param-
metric communication model that relates interacting services. Differently
2.4. DISCUSSION

to other existing approaches, this model allows for the representation of a wide range of communication protocols and mechanisms with potentially unbounded queues.

**Analysis techniques.** We adapted model checking techniques, and in particular symbolic model checking, in order to perform automated analysis for the following reasons. First, there exists a wide range of implementations that allow for efficient analysis of large-scale problems. Second, these techniques enable different kinds of analysis, including verification of temporal properties and simulation checking. Third, it can be used as a platform for integration and implementation of more advanced analysis techniques, that may be specific to a certain problem domain. In our work we exploit this capability in order to develop the algorithms and methods to deal with the specific features of SOA and Web services domain.

In particular, we proposed an approach that allows one to reason on communication models in service compositions, which was inspired by the synchronizability analysis of [56], and various techniques for reasoning on communication channels [2].

We also exploited abstraction-based reasoning to deal with infinite and complex data-flow. These techniques were extended in order to manage information incompleteness.

In a similar way we investigated the timed analysis in application to Web services. The formalisation and algorithms were adopted to focus on “business” timeliness of composition behavior, to reason on transactional aspects and to extract their quantitative properties.
Chapter 3

Formal Model

In this chapter we describe a formal model that allows for the representation of the Web service compositions. The model defines a single service as state transition system of a special form suitable for describing the service behavior, the data flow, and the time flow. A formal model for the composition interconnects the services using a parametric communication model, which is used to represent asynchronous message exchanges in a wide range of protocols. We also define the execution semantics of the formalism, needed to carry out behavioral analysis tasks, and discuss the assumptions we made on the form of the compositions and services.

3.1 Conceptual Model of the Composition

In the previous chapter we discussed the Web service technology and presented a generic conceptual model of a service composition. The composition consists of a set of participants that interact with each other through asynchronous message exchange. Both software services and human users may participate in the composition execution. We do not distinguish between these types, and assume that each participant is provided with a public specification that in some way defines its functionality and behavior, e.g., WSDL and BPEL documents.
The message exchange is performed on top of the low-level communication systems, or middleware, that are responsible for message delivery and storage, potentially having various implementations and running various transport protocols. The way the communication systems are implemented and represented is the key aspect for the analysis of service compositions. Indeed, the behavior of the application may depend on the ordering of concurrent message events, which in turn is up to the implementation of queuing system, i.e., routing of messages, their storage, ordering rules, etc. The Web service technology, however, does not commit to a particular implementation and protocol, and abstracts away this information from the composition specification.

We describe these asynchronous, complex, and heterogeneous messaging systems using a formal communication model, which is parametric with respect to the number of queues, their ordering, bounds, and message alphabets. This parametrization, together with the special analysis techniques presented in this thesis, allow us to consider and manage the following important aspects of the Web service interactions:

- Invoke operations are non-blocking. Due to asynchronous, loosely coupled nature of Web services, the message emission cannot be blocked even if the receiver is not ready to accept the message.

- Queue are unbounded, but with the restriction that the number of messages in the queues cannot grow unboundedly. That is, we do not define a limit to length of the queues “a priori”, however we consider invalid all those composite systems where the number of messages in the queue can grow unboundedly. (This corresponds to assume that the queues are “long enough” to contain all the messages that need to be stored in the executions of the systems.)

- A wide range of middleware implementations and protocols is allowed.
That is, we can not commit to a specific configuration, but should consider the behavior of the composition under various settings. In order to reflect this property, we assume the most situation, where the messages can be consumed in any arbitrary order, regardless the order in which they are stored in the queues. Assuming the most general behavior of queues allows us to guarantee that the theoretical model includes all behaviors that are possible in any specific engine.

The ability to analyze Web service compositions relies on certain hypotheses on the formal model. First, these hypotheses will allow us to abstract from low-level technological issues that are irrelevant for the “logic” of the composition, leading to a more compact representation of the composition and to a simpler analysis process. In other words, we focus on the functional aspects of the composition, and ignore the problems related to the non-functional ones, such as the reliability of interactions, or the service performance. Second, these assumptions will allow for the simplified formalization of the composition model, and, consequently, will enable more efficient analysis algorithms.

**H1** The communication channels used by the participants are disjoint. That is, it is never the case that two services are able to send/receive the same message to/from a third service. Formally, this means that the sets of messages used by pairs of services are disjoint, and that the end points of the communications are statically fixed.

**H2** The number of services participating to a composition is fixed, the participants cannot be created or destroyed dynamically.

**H3** The communication channels are perfect, i.e. no message losses can ever occur. This assumption may be enforced by special techniques in the WS domain, such as WS Reliable Messaging [107] or monitoring.
3.2 FORMAL MODEL OF WEB SERVICE

H4 Representing time aspects in the composition behavior, we consider only those times that are critical from the point of the business logic, i.e., the time required by the participating actor to carry out long-running tasks and take their decisions. Instead, the times, required, for instance, by the communications, are orders of magnitude smaller and can be neglected.

3.2 Formal Model of Web Service

The formal model we use as a basis for the required analysis techniques consists of three parts, namely the data model, the time model, and the behavioral model. The data model provides a formalisation of the data manipulated by the services and is used to reason on the data flow of the compositions. The time model is used to represent the flow of time during the execution of the service. The control flow is defined by the behavioral model, used to represent a behavior of the service.

3.2.1 Data Model

We represent data, operations on data, and data flow of the system execution using a ground model. We further explain the definition of the STS with regards to the ground context.

Definition 3.1 (Ground Context)

A ground context is a tuple \( \langle T, V, F \rangle \), where

- \( T \) is a set of infinite or enumerative types;
- \( V \) is a set of typed variables;
- \( F \) is a set of typed functions;
We denote the type of the variable $x$ (respectively, function $f$) as $T(x)$ (resp., $T(f)$).

A ground state is characterized by a complete assignment over the set of typed variables $V$ and by the function interpreter $I_f$.

**Definition 3.2 (Ground State)** Given a ground context $(T, V, F)$, we define a ground state $g$ as a pair $(I_v, I_f)$ where

- $I_v$ is a set of pairs $\langle x, v \rangle$ such that for all $x \in V$ there exists a unique $\langle x, v \rangle \in I_v$ with $v \in T(x)$;
- $I_f$ is the function that given a typed function $f \in F$ and a set of values $v_1, \ldots, v_n$, with $n \geq 0$, returns the result of the computation of $f(v_1, \ldots, v_n)$.

We distinguish functions with fixed interpretation $F^s \subseteq F$, i.e.,

$$\forall f \in F^s, \forall I_f, I'_f, \forall v_1, \ldots, v_n, I_f(f, v_1, \ldots, v_n) = I'_f(f, v_1, \ldots, v_n);$$

(typed) constants $F^c \subseteq F^s$ as functions with zero parameters, so that

$$\forall f_1, f_2 \in F^c, \forall I_f \text{ if } T(f_1) = T(f_2) \text{ then } I_f(f_1) \neq I_f(f_2);$$

and functions with arbitrary interpretation $F^u = F \setminus F^s$. Note that two different applications of some function $f \in F^u$ may produce two different values even for the same input in different ground states. These functions are used for the non-deterministic modelling of internal service calculations that depend on some “hidden” information.

Let $e$ denote an expression and $E$ a set of expressions. We use $t$ to denote a term, and $T$ to denote a set of terms. We also write $x$ for a variable in $V$. The syntax of expression is as follows\footnote{Analogously, predicates may also be interpreted as boolean functions.}:

- $E \equiv (t_1 = t_2) \mid \neg e \mid (e_1 \lor e_2)$ that is equality between terms, negation or disjunction of expressions;
3.2. FORMAL MODEL OF WEB SERVICE

- $T \equiv x \mid f(t_1, \ldots, t_n)$ that is a variable or a function call.

A data condition $\phi$ is an expression of the form presented above. A set of all data conditions is denoted as $\Phi_D$. A data assignment $\omega$ has the form $(x := t)$. A set of assignments is denoted as $\Omega$.

Definition 3.3 (Evaluation Function) We define the evaluation function $\Gamma_g$ as the function that given a term $t$ or an expression $e$ returns the result of the computation with respect to a ground state $g = \langle \mathcal{I}_v, \mathcal{I}_f \rangle$:

- $\Gamma_g(x) = v$, where $x$ is a variable and $\langle x, v \rangle \in g$;
- $\Gamma_g(f(t_1, \ldots, t_n)) = v$, where $v = \mathcal{I}_f(f, \Gamma_g(t_1), \ldots, \Gamma_g(t_n))$;
- $\Gamma_g(t_1 = t_2) = true$ iff $\Gamma_g(t_1) = \Gamma_g(t_2)$;
- $\Gamma_g(\neg e) = \neg \Gamma_g(e)$;
- $\Gamma_g(e_1 \lor e_2) = \Gamma_g(e_1) \lor \Gamma_g(e_2)$.

We say that the ground state satisfies a formula $\phi \in \Phi_D$, written as $g \models \phi$, iff $\Gamma_g(\phi) = true$.

Definition 3.4 (Ground Update) The update of a ground state $g$ with an assignment $\omega = (x := t)$, denoted as $\text{update}(g, \omega)$, is the state $g'$, s.t.

$$\forall x' \in \mathcal{V}, \; \Gamma_g'(x') = \begin{cases} \Gamma_g(x'), & \text{if } x' \neq x \\ \Gamma_g(t), & \text{if } x' = x \end{cases}$$

3.2.2 Time Model

In order to model timed behavior of the compositions, we adapt the formalism of timed automata for capturing the aspects specific to the Web service domain. For these reasons, the service model is equipped with a set of special variables, namely clock variables. The values of these variables synchronously increase with the passing of time.
CHAPTER 3. FORMAL MODEL

Let $\mathcal{X}$ be a set of clock variables. The syntax of time constraints $\phi \in \Phi_T$ on the clock values has the form $true \ | \ c \sim t \ | \ \phi_1 \land \phi_2$, where the operator $\sim$ is one of $\{\leq, <, \neq, =, \geq, >\}$, and $c$ is an element of a domain of time values $\mathbb{T}$. A reset $r \in Y$ has the form $(x := 0)$, that is the value of clock $x$ is set to zero.

A clock valuation is a function $u : \mathcal{X} \to \mathbb{T}$ from the set of clocks to the domain of time values. We denote a set of all clock valuations as $\mathbb{T}_C$, and write $u_0(x) = 0$ for all $x \in \mathcal{X}$ to denote the initial clock valuation. We say that the clock valuation satisfies a constraint $\phi \in \Phi_T$, written as $u \models \phi$, iff the constraint evaluates to true under the valuation.

3.2.3 STS Formalism

We encode the behavior of a Web service as a state transition system (STS). A STS describes dynamic systems that can be in one of their possible states (some of which are marked as initial states) and can evolve to new states as a result of performing some actions. This evolution is defined by the transition relation. The relation defines conditions, under which the action can be performed, and effects of these executions, which specify the modification of the service states.

The state of a STS is defined by the location that specifies the execution point of the service control flow, and by the memory that specifies the current values of the local service variables $\mathcal{V}$. The behavioral specification, therefore, defines how the actions change the service locations, and what the rules for memory updates are. The set of locations is denoted as $\mathcal{S}$.

Following the standard approach in process algebras, we distinguish external actions representing service interactions, and internal actions, which are used to represent evolutions of the system that do not involve interactions with the external services. We denote an internal action as $\tau$. External actions are distinguished in input actions $\mathcal{I}$, which represent the
reception of messages, and output actions $O$, which represent the emission of messages to external services. The send action is denoted as $\overrightarrow{\mu}(\bar{x})$, where $\mu$ is a service operation, or message type, and $\bar{x}$ is a vector of service variables, from which the message content is populated. The receive operation is denoted as $\overleftarrow{\mu}(\bar{x})$, where the message content is assigned to the variables in $\bar{x}$. The set of all service actions is denoted as $A$.

The timed part of STS formalism is defined as follows. Each transition may be declared as either urgent or non-urgent. The urgent transitions are fired as soon as they are enabled, and have a priority over non-urgent transitions. The non-urgent transitions are used to model time-consuming task, while, by default, modelling the Web service activities as instant using urgency declaration. A transition is also equipped with a timed transition guard that have a form of the clock constraints. In order to ensure progress, some locations may be equipped with the time invariants that constrain the time that may be spent in the location. Finally, a set of clocks may be reset when a particular transition is performed.

**Definition 3.5 (STS)**

A state transition system is a tuple $\langle S, S_0, V, X, A, R, L_I \rangle$, where

- $S$ is the set of locations and $S_0 \subseteq S$ is the set of initial locations;
- $V$ is a set of local variables and $X$ is a set of service clocks;
- $A$ is a set of actions;
- $R \subseteq S \times \Phi \times A \times \Omega^* \times 2^X \times \{true, false\} \times S$ is the transition relation;
- $L_I : S \rightarrow \Phi_T$ are the functions assigning time invariants to locations.

A transition $t = (s, \phi, a, \Omega, Y, \vartheta, s') \in R$ changes the location from $s$ to $s'$, fires an action $a \in A$, makes a set of assignments $\Omega$ and resets subset of clocks $Y$ to zero. If $\vartheta = true$ the transition $t$ is urgent. The transition guard has the form $\phi \in \Phi_D \cup \Phi_T$.

48
3.3 Formal Model of Web Service Composition

We now give a formal model and semantics of a Web service composition. In this model the composition is represented as a network of STSs corresponding to the participating services that interact with each other through a communication medium, referenced to as communication model. This communication model is given by queues that store the messages from and to the partners.

3.3.1 Modelling Composition Structure

The composition model is built from \( n \) STSs \( \langle S^i, S^i_0, V^i, X^i, A^i, R^i, L^i_i \rangle \) representing the participating Web services, and a communication model \( \Delta \), i.e., a set of queues that store the messages exchanged among partners. A global state of the composition at a particular execution point is defined by the locations of the participants, by the values of their variables and clocks, and by the content of message queues. We call such a description a configuration of the composition.

More formally, a configuration \( \gamma \) of the composition is a tuple \( \langle \bar{s}, \bar{g}, C, \bar{u} \rangle \), where \( \bar{s} \) is a global location, i.e., a combination of the locations of all the participants; \( \bar{g} \) is a global ground state that represent the values of all the local data variables of the participants \( V = \bigcup_i V^i \); \( C \) represents the queue contents; and \( \bar{u} \) is a global clock valuation that represents the values of all the local clocks \( X = \bigcup_i X^i \). A global location is a vector \( \bar{s} = \langle s_1, \ldots, s_n \rangle \), where \( s_i \) is a location of the \( i^{th} \) STS. We denote a vector with component \( s_i \) updated to \( s_i' \) as \( \bar{s}[s_i'/s_i] \).

3.3.2 Modelling Message Interactions

The component STSs interact with each other sending and receiving messages. A message \( \alpha \) is identified by the message type \( \text{OP}(\alpha) \) and its content
VAL(α) (i.e., data exchanged) as a vector of data values. We denote a set of all messages as $M$, and a set of all message types as OP($M$).

The communications among STSs are performed through a set of $m > 0$ queues with disjoint alphabets $M_j \subseteq M$ with $1 \leq j \leq m$. A queue $q_j$ may be declared as bounded, with the corresponding capacity $0 < b_j < \infty$, or unbounded, in which case $b_j = \infty$. As one can see, given the same set of STSs, different configurations may be used to represent their composition. These configurations are parametric with respect to the number of queues, to the distribution of the queue alphabets, to the ordering of messages inside the queues, and to the queue bounds. We denote such configurations as communication models.

**Definition 3.6** A communication model for the STS composition is a tuple $\Delta = \langle B, L_M, L_O \rangle$, where $B = \langle b_1, \ldots, b_m \rangle$, is a vector of queue bounds, $L_M : M \rightarrow [1 \ldots m]$ is a function that associates a message $\alpha$ with a queue $i$, and $L_O : [1 \ldots m] \rightarrow \{\top, \bot\}$ is a function that declares the queue as either ordered or unordered. The alphabet $M_i$ of queue $i$ is defined as $M_i = \{\alpha \mid L_M(\alpha) = i\}$.

The state of the queues in a particular moment of time is defined by the queue structure, which represents the distribution of messages among queues, and by the queue content, which is represented by the content of the stored messages.

More precisely, let $M^*$ be a set of sequences (or strings) of elements from $M$. Let also $N^M$ be a set of multisets of $M$, i.e. set of mappings from $M$ to natural number $\mathbb{N}$. We define the queue structure as the vector $C = \langle w_1, \ldots, w_m \rangle$, where $w_j \in M_j^*$ and $j^{th}$ queue is ordered, or $w_j \in N^M$ and $j^{th}$ queue is unordered.

Given two elements $w$ and $w'$, we write $w.w'$ to denote the string concatenation, if $w, w' \in M^*$, and multiset union, if $w, w' \in N^M$. We extends
this operator to the queue content as follows: \( C.\alpha = \langle w'_1, \ldots, w'_m \rangle \), where
\( w'_j = w_j.\alpha \) if \( \alpha \in M_j \), and \( w'_j = w_j \) otherwise.

We also write \( w \leq w' \), if \( w, w' \in M^* \) and \( w \) is a prefix of \( w' \): \( w' = w.w'' \).
We write \( w \leq w' \), if \( w, w' \in \mathbb{N}^M \) and \( w \) is a submultiset of \( w' \): \( \forall \alpha \in M, w(\alpha) \leq w'(\alpha) \). Analogously, \( C \leq C' \) if for any queue index \( i \) \( w_i \leq w'_i \).
We write \( |C| \leq B \) to specify that \( |q_i| \leq b_i \).

When an external transition is performed, the data is passed to and from the queues. That is, when an output action \( \mu(\bar{x}) \) is performed, the message content is populated from the values of the variables \( \bar{x} \) in the current configuration \( \bar{g} \). In other words, a message \( \alpha \) is produced such that \( \text{OP}(\alpha) = \mu \), and \( \text{VAL}(\alpha) = \Gamma_{\bar{g}}(\bar{x}) \). The message is added to the corresponding queue, but only if the queue is not full: \( C' = C.\alpha \land |C'| \leq B \).

Analogously, an input action \( \mu(\bar{x}) \) is applied to a message \( \alpha \), such that \( \mu = \text{OP}(\alpha) \) and the variables in \( \bar{x} \) will have the values from the message content: \( \Gamma_{\bar{g}}(\bar{x}) = \text{VAL}(\alpha) \). The action is possible if there is a corresponding message stored in the queue: \( C = \alpha.C' \).

### 3.3.3 Global Transition System

The formal model of a Web service composition is defined as a global transition system (GTS). This model defines how the composition evolves, how the variables and message queues are modified, and how time increments.

The data flow of the composition is defined by the assignments, message passing, and data conditions on transitions. That is, the assignment and message passing defines the modification of the global ground state and the queue contents in the current configuration.

The timed behavior of the compositions is based on the formalism of timed automata. In particular, the fact that the operation takes a certain amount of time is represented by time increment in the state, followed by the immediate execution of the operation. The time can pass only in
the states where there are no urgent transitions enabled. The location invariants and time guards are used to ensure the (correct) progress of the composition execution. The location invariants should be true when the system is in the corresponding location. The global location invariants are denoted as $L_I(s) = \bigwedge_i L_{I_i}(s_i)$.

We call all possible local actions a set of the composition actions, written as $A = \bigcup_i A_i$. We also write $\langle \bar{s}, \bar{g}, C, \bar{u} \rangle |\phi$ to denote that the configuration satisfies $\phi$.

Some local transition of a participant can be performed if the transition is applicable in a configuration: the transition guard should be true, the input/output is allowed. The resulting configuration is defined as an effect of performing the transition: the assignments are performed, the clocks are reset, the queue content is possibly updated.

**Definition 3.7 (Applicability and Effect)**

A local transition $t = (s_i, \phi, a, \Omega, Y, \vartheta, s_i') \in R_i$ is applicable in a configuration $\gamma = \langle \bar{s}, \bar{g}, C, \bar{u} \rangle$, written as $\text{applicable}[\gamma](t)$, if and only if

- $\bar{g}, \bar{u} \models \phi$;
- if $a = -\rightarrow \mu(\bar{x})$ then $\forall \alpha \in M$ if $\text{op}(\alpha) = \mu$ then $|C.\alpha| \leq B$;
- if $a = -\leftarrow \mu(\bar{x})$ then $\exists C', \exists \alpha \in M$ such that $C = \alpha.C'$ and $\text{op}(\alpha) = \mu$;

The effect of performing the transition $t$ in the configuration $\gamma$ is a configuration $\gamma' = \langle \bar{s}', \bar{g}', C', \bar{u}' \rangle$, written as $\gamma' = \text{exec}[\gamma](t)$, such that

- $\bar{u}' = \bar{u}[Y \mapsto 0]$, $\bar{g}' = \text{update}(\bar{g}, \Omega)$, and $\bar{s}' = \bar{s}[s'/s_i]$;
- if $a = -\rightarrow \mu(\bar{x})$ then $\exists \alpha \in M$ s.t. $\text{op}(\alpha) = \mu$, $\text{val}(\alpha) = \Gamma_{\bar{g}}(\bar{x})$, $C' = C.\alpha$;
- if $a = -\leftarrow \mu(\bar{x})$ then $\exists \alpha \in M$ s.t. $\text{op}(\alpha) = \mu$, $\Gamma_{\bar{g}'}(\bar{x}) = \text{val}(\alpha)$, $C = \alpha.C'$;
- if $a = \tau$ then $C' = C$. 

52
Definition 3.8 (Global Transition System)
A Global Transition System (GTS) that represents the composition of \( n \) STSs under a communication model \( \Delta = \langle B, L_M, L_O \rangle \) is a tuple \( \Sigma_\Delta = (\Gamma, \Gamma_0, \leadsto) \), where \( \Gamma \) is a set of configurations, \( \Gamma_0 \subseteq \Gamma \) is a set of initial configurations with \( \bar{u} = \bar{u}_0 \), and \( \leadsto \subseteq \Gamma \times \{ \mathcal{A} \cup \text{TICK} \} \times \Gamma \) is a transition relation such that:

\[
\begin{align*}
\bullet & \quad (\langle \bar{s}, \bar{g}, C, \bar{u} \rangle, \text{tick}, \langle \bar{s}, \bar{g}, C, \bar{u} + d \rangle) \in \leadsto, \text{ if } \\
& \quad - \forall 1 \leq i \leq n, \forall t \in R^i, \text{ if applicable}[\langle \bar{s}, \bar{g}, C, \bar{u} \rangle](t) \text{ then } \vartheta = \text{false}; \\
& \quad - (\bar{u} + d) \models L_I(\bar{s}); \\
\bullet & \quad (\langle \bar{s}, \bar{g}, C, \bar{u} \rangle, a, \langle \bar{s}', \bar{g}', C', \bar{u}' \rangle) \in \leadsto, \text{ if } \exists 1 \leq i \leq n, \exists t \in R^i \text{ such that } \\
& \quad - \text{applicable}[\langle \bar{s}, \bar{g}, C, \bar{u} \rangle](t); \\
& \quad - \langle \bar{s}', \bar{g}', C', \bar{u}' \rangle = \text{exec}[\langle \bar{s}, \bar{g}, C, \bar{u} \rangle](t); \\
& \quad - \bar{u}' \models L_I(\bar{s}').
\end{align*}
\]

That is, either there are no urgent transitions enabled, the system remains in the same state and time passes, or a certain transition of some STS immediately takes place.

3.3.4 Composition Behavior

The behavior of GTS can be described with a set of directed (possibly infinite) labeled trees, called reachability trees \( RT \). Nodes in such tree are labeled with (reachable) configurations \( \gamma \in \Gamma \); the root is labeled with one of the initial global configuration \( \gamma_0 \in \Gamma_0 \); edges are labeled with actions \( a \in \mathcal{A} \). The reachability graph \( RG \) is obtained from \( RT \) by merging nodes labeled with identical global states.

We say that action \( a \in \mathcal{A} \) is fireable in a configuration \( \gamma \), if there is a transition \( (\gamma, a, \gamma') \in \leadsto \). In this case, we write \( \gamma \xrightarrow{a} \gamma' \). Let \( \pi = \)}
Let $\gamma_1, a_1, \gamma_2, a_2, \ldots$ be a (possibly infinite) sequence of configurations and actions interleaved. We say that the sequence is fireable from $\gamma_1$, written as $\gamma_1 \xrightarrow{\pi}^*, \text{if } \forall k \geq 1, \gamma_k \xrightarrow{a_k} \gamma_{k+1}$.

We remark that the composition configuration defines not only the local states of the services and exchanged messages, but also the state of the infrastructure, i.e., the queue content. The latter may vary for different middleware and therefore is not important for the verification of the Web service composition. We define the behavior of the composition omitting the information about queue content.

**Definition 3.9 (Composition behavior)**

Given a configuration $\gamma = \langle \bar{s}, \bar{g}, C, \bar{u} \rangle$, we call observable state $\sigma(\gamma)$ a tuple $\langle \bar{s}, \bar{g}, \bar{u} \rangle$, representing a control, data, and time valuation.

We call a sequence $\omega = \sigma_0, a_0, \sigma_1, a_1, \ldots$ a run of the composition if there is a sequence $\pi = \gamma_0, a_0, \gamma_1, a_1, \ldots$ s.t. $\gamma_0 \xrightarrow{\pi}^*, \gamma_0 \in \Gamma_0$, and for each $i \geq 0$, $\sigma_i = \sigma(\gamma_i)$. We denote that the run $\omega$ is fireable from a state $\sigma_0$ as $\sigma_0 \xrightarrow{\omega}^*$. A set of runs is called a behavior of the composition:

$$\Xi = \{ \omega \mid \sigma_0 \xrightarrow{\omega}^*, \sigma_0 = \sigma(\gamma_0) \text{ for some } \gamma_0 \in \Gamma_0 \}$$

The behavior of the Web service composition defines the control and data flow of the composition evolution. It also defines the valuations of timers during the composition execution. Note, that the compositions based on different implementation of the communication infrastructure (i.e., communication models) may generate the same composition behavior.

### 3.3.5 Infiniteness of the GTS Model

A natural question that arises with the given formalizations is whether we can find a way to automatically analyze a Web service composition specification. Unfortunately, the answer is no. The presented formalism of
STSs and their composition is too expressive for the direct application of formal methods for the analysis purposes. Indeed, it is easy to see that the presented model is Turing-powerful, thus making the verification problem undecidable. This undecidability has several essential sources.

First, the decidability is prohibited by the complexity of the data model used in the formalism. Indeed, the manipulation of infinite data domains and arbitrary functions in data assignments and conditions, implies that the analysis is in general undecidable, even for timeless models and bounded queues.

Second, the complexity of the communication model may also lead to the undecidability results even if the data and time are abstracted away. The definition of the composition depends on the communication model. If the queues are unbounded, for a wide range of models the analysis questions become undecidable. In particular, in [19, 112] it was shown that the analysis is undecidable for the formalism of communicating finite-state machines. Many positive results obtained for particular classes of such systems [75, 1, 17, 51], are discarded when the data flow is considered [2].

In order to be able to carry out the analysis tasks for the compositions represented in our formal model, the suitable techniques and methods to avoid undecidability are of topmost importance. The following chapters are dedicated to such methods, allowing us to provide a basis for the formal analysis framework for Web service compositions.

### 3.4 Discussion

We presented a formal model suitable for the representation of Web service compositions. The model focuses on the ability to represent asynchronous communications in various queueing systems, to describe time-related and data-related properties of the composition behavior.
The presented formalism is particularly suitable for the bottom-up development of the service composition, where the composite system is constructed from pre-existing service models, specified, e.g., in WSDL/BPEL documents. However, this model may be used for supporting a top-down approach, where a global model is designed and then used as a blueprint document for the local implementations. In [83] we used this model as a reference model of the composition implementation for checking its conformance to a choreography specification. Analogously, in [82] it was applied for the analysis of various notions of the choreography realizability.

Our formal model was inspired by several approaches to the formal analysis of software systems. The model of asynchronous composition is closely related to the formalism of Mealy machines widely used in literature for the analysis of interactions in distributed systems, and in particular in service compositions [56]. Our model differs in being parametric, which allows for more flexible representation and analysis of various scenarios and systems, and in a situation, when the real model is not known a priori.

Our representation of a service as a state-transition system is close to some other works on the composition analysis. In particular, the transition system enriched with variables, transition conditions and assignments is used in [56, 101]. Our model, however, extends this model with arbitrary functions. In this way the formalism supports advanced features of Web service data descriptions, such as modelling of non-deterministic data assignments and complex XML operations.

The timed semantics of the service behavior presented here in a high degree reflects the formalism of timed automata (with urgency). We adapt this formalism in order to focus on the aspects important for business logic. For these purposes, the execution is instant by default, and all the critical activities that require certain “business” time should be explicitly modelled, using, e.g., requirements presented in Chapter 7.
Chapter 4

Representing Web Service Specifications

In this chapter we describe how the Web service composition specification and the corresponding behavioral requirements are formalized. We show, how a WSDL/BPEL specification defining a single participant to the composition may be represented in the STS formalism. We also present what the behavioral requirements are, introduce a formal language for the specification of such requirements, and define its semantics with respect to the behavior of the Web service composition formalism.

4.1 Modelling BPEL Processes

The Business Process Language for Web Services (WS-BPEL, [5]) is a standard for specifying and executing business processes that use Web service interfaces. It provides a core of process description concepts needed for the definition of stateful, synchronous and asynchronous, long-running interactions among distributed applications.

This core of concepts is used both for the definition of the internal business processes of a participant to an interaction and for describing and publishing the external business protocol that defines the interaction be-
4.1. MODELLING BPEL PROCESSES

behavior of a participant without revealing its internal implementation. The language allows for specifying a reusable process definition that can be exploited in different ways and in different scenarios, providing an expressive power for the execution, specification, and analysis of the Web service compositions.

4.1.1 Overview of BPEL

A BPEL document describes a particular business process. The process definition consists of several parts describing partner links, process variables, correlation sets, main process workflow, the fault and compensation handling activities.

The partner link declarations are used to define the relation between the process and its partners. It defines the role of the process in this relation (consumer or provider of an interface), and the interfaces used/provided by that role. The interfaces, operations, as well as their parameters and types, are specified in the corresponding WSDL documents.

The process variables are used to represent the state of the business process, they contain the information received from or sent to the partners of the process. The variables may be of primitive data types (e.g., strings, boolean, integers) or of some complex types defined in a WSDL document.

The correlation sets define the parts of message data that are used to associate and route a particular message to a particular instance of the business process. Such information tokens uniquely identify the instance of the business process.

The process flow is defined by a set of process activities. They specify the operations to be performed, their ordering, conditional logic, reactive rules, etc. We distinguish the following groups of activities: basic activities, structured activities, and the specific operational blocks, namely fault and compensation handlers.
Basic activities represent primitive operations performed by the process, such as message emission/reception (*invoke*, *receive*, and *reply* activities), data modification (*assign*), process termination (*terminate*), waiting for a certain period of time (*wait*), or doing nothing (*empty*).

Structured activities define the order in which a collection of activities occurs. They compose the basic activities into structures that express the control flow patterns. The structured BPEL activities include *sequence*, *switch*, and *while* that model traditional control constructs; *pick* that models nondeterministic choice based on external events (i.e., message reception or timeout); *flow* activity that models parallel execution of the nested activities. The structured activities can be recursively nested and combined.

Fault handling in BPEL is thought of as a mode switch from the normal processing. It is interpreted as “reverse work”, since it aims at undoing the unsuccessful work. The fault may arise on reception of the fault message, or on explicit invocation of the *throw* activity. The fault handler declaration specify the activities to be performed when a fault arises.

The compensation handlers are used to reverse the effect of some unit of work that has completed with a fault. The compensation is always initiated within a fault handler, and may also require a compensation of some nested, previously successful, activities. A compensation handler is always associated with a work unit (BPEL scope), and is invoked (explicitly or implicitly) using the BPEL *compensate* activity.

**Example 4.1** Let us consider a BPEL process that describes the Flight Reservation Service, a part of the Virtual Travel Agency case study that we exploit in the following chapters. The process workflow is represented in Fig. 4.1.a. The process starts when the flight request message is received. The availability of the flight for the date and the city specified in the request is checked. If a corresponding flight exists, an offer is prepared
4.1. MODELLING BPEL PROCESSES

Figure 4.1: Example of a Flight Reservation BPEL process
and returned as an outcome of the request operation. Otherwise, a fault message is returned and the process terminates. After sending an offer, the process waits for positive or negative acknowledgement from the customer. In the former case, the ticket is prepared and send to the customer in reply to the positive acknowledgement.

In Fig. 4.1.b (a part of) the WSDL document that defines the Flight service interface is presented. In particular, the document declares a message type named “requestMsg” that consists of two parts, namely “date” and “city”. A port type declaration specifies the set of operations that the service can perform. For example, the “Flight_PT” port types declares the “request” operation that accepts a message of type “requestMsg” as input, returns a message of type “offerMsg” as output, and may throw a fault named “not_avail” of type “NAMsg”. A partner link type “Flight_PL” defines a relation between the service and its user, where the role “provider” is associated to the port type “Flight_PT”.

Figure 4.1.c represents (a part of) the BPEL of the Flight Reservation process. The “Flight_PL” partner link states that BPEL process plays a role of “provider” in the “Flight_PL” relation. The “req” variable is used to store the request received from the user, while boolean variable “available” is used to determine availability of the requested ticket. The behavior of the process reflects the graphical representation: upon the reception of the request (receive activity) and depending on the availability of the tickets (switch), either an offer is prepared and sent to the user (reply), or a fault message is returned and the process terminates (reply with a fault).

4.1.2 From BPEL to STS

We model BPEL specification using the State Transition System formalism, that is, each process is represented as a separate STS. The declarative
part of the BPEL document is used to define the context of the process, i.e.,
the input/output actions of the process, data variables, etc. In particular,
the partner links, together with the corresponding port types, operations,
and messages are used to uniquely define the operations of the process. The
declaration of the process variables are used to define the variables of
the corresponding STS. Without loss of generality we assume that these
declarations are mapped into a finite set $V$ of variables of certain types
with potentially infinite domains. More detailed description of the XML
data manipulation, and on the representation of XPath functions and ex-
pressions may be found in [57].

**Example 4.2** Consider a specification of the Flight Reservation process
represented in Fig. 4.1. For the operation “request” we define three message
types, namely request, reply_request, and notavail_request. The variable
“req” of type “requestMsg” is represented as two variables of simple types,
namely req_date of type date and req_city of type string.

The translation of the process workflow consists of the recursive trans-
lation of the process activities. Below we give an intuitive and informal
description of the translation of basic activities, structured activities, time-
related activities, fault and compensation handlers.

We use the following notations to represent this translation. We denote
the location, where the activity starts, as $s_b$, and the location, where the
activity (normally) ends, as $s_e$. Note that an activity may have several
final locations. We write $s_b \xrightarrow{a} s_e$ to denote the (recursive) mapping of the
(structured or basic) activity $a$ with $s_e$ and $s_b$ being an initial and final
locations of the activity. In the description of the mapping we specify the
explored locations, transitions, actions, and their parameters.

\footnote{For the sake of simplicity we assume that the operation names are not shared among the interfaces and are sufficient for operation identification.}
### Table 4.1: Mapping basic BPEL activities to STS

<table>
<thead>
<tr>
<th>Activity</th>
<th>Example</th>
<th>STS representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>receive</td>
<td>variable=&quot;var&quot; operation=&quot;op&quot;</td>
<td>(s_b, true, $\overrightarrow{op}(var)$, $\emptyset$, $\emptyset$, true, s_e) $\in$ $\mathcal{R}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$s_b, s_e \in \mathcal{S}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\overrightarrow{op}(var) \in \mathcal{I}$</td>
</tr>
<tr>
<td>reply</td>
<td>variable=&quot;var&quot; operation=&quot;op&quot;</td>
<td>(s_b, true, $\overrightarrow{reply \cdot op}(var)$, $\emptyset$, $\emptyset$, true, s_e) $\in$ $\mathcal{R}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$s_b, s_e \in \mathcal{S}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\overrightarrow{reply \cdot op}(var) \in \mathcal{O}$</td>
</tr>
<tr>
<td>invoke</td>
<td>inputVariable=&quot;ivar&quot; operation=&quot;op&quot;</td>
<td>(s_b, true, $\overrightarrow{op}(ivar)$, $\emptyset$, $\emptyset$, true, s_e) $\in$ $\mathcal{R}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$s_b, s_e \in \mathcal{S}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\overrightarrow{op}(ivar) \in \mathcal{O}$</td>
</tr>
<tr>
<td>invoke</td>
<td>inputVariable=&quot;ivar&quot; outputVariable=&quot;ovar&quot; operation=&quot;op&quot;</td>
<td>(s_b, true, $\overrightarrow{op}(ivar)$, $\emptyset$, $\emptyset$, true, s_e) $\in$ $\mathcal{R}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$s_b, s_e \in \mathcal{S}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\overrightarrow{op}(ivar) \in \mathcal{O}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(s_b, true, $\overrightarrow{f1 \cdot op}(fv1)$, $\emptyset$, $\emptyset$, true, s_{f1}) $\in$ $\mathcal{R}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(s_b, true, $\overrightarrow{f2 \cdot op}(fv2)$, $\emptyset$, $\emptyset$, true, s_{f2}) $\in$ $\mathcal{R}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s_b, s_e, s_{f1}, s_{f2} $\in$ $\mathcal{S}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\overrightarrow{op}(ivar) \in \mathcal{O}$, $\overrightarrow{op}(ovar) \in \mathcal{I}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\overrightarrow{f1 \cdot op}(fv1) \in \mathcal{I}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\overrightarrow{f2 \cdot op}(fv2) \in \mathcal{I}$</td>
</tr>
<tr>
<td>assign</td>
<td>copy from expression=&quot;term&quot; to variable=&quot;var&quot;</td>
<td>(s_b, true, $\tau$, {var := term}, $\emptyset$, true, s_e) $\in$ $\mathcal{R}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s_b, s_e $\in$ $\mathcal{S}$</td>
</tr>
<tr>
<td>empty</td>
<td></td>
<td>(s_b, true, $\tau$, $\emptyset$, $\emptyset$, true, s_e) $\in$ $\mathcal{R}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s_b, s_e $\in$ $\mathcal{S}$</td>
</tr>
<tr>
<td>terminate</td>
<td></td>
<td>(s_b, true, $\tau$, $\emptyset$, $\emptyset$, true, s_F) $\in$ $\mathcal{R}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s_b, s_F $\in$ $\mathcal{S}$</td>
</tr>
</tbody>
</table>

### Mapping Basic Activities

The examples of the basic activities and the corresponding STS transitions are represented in Table 4.1. The interaction activities are described with the receive/reply/invoke activities. The receive activity defines the recep-
tion of the message. It specifies an operation, an interface, and a variable to be populated from the message content.

The *reply* activity defines the message being sent to the partner in reply to request-response invocation. It also specifies an operation, an interface, while the variable is used to specify the value of the message to be sent. When the response to the request indicates some fault, the optional *fault-Name* attribute is present and the output action has the form $\text{fn}_\text{op}(\text{var})$, where $\text{fn}$ is the name of the corresponding fault.

The *invoke* activity defines a one-way or a request-response invocation of the partner. In the first form the activity defines a message emission without waiting for a response. The following information is specified: an operation and an interface of the invoked partner, variable used to specify the transmitted value. The second form of the *invoke* activity is used to define an invocation followed by the reply from the partner (two-way invocation). Additionally, the output variable is specified to store the value of the received message in case of normal completion of the activity, and one or more fault variables associated to the particular faults in case of an error returned by the partner. The message emission part is mapped to the corresponding output transition, followed by the reception transitions, one for each possible outcome.

There are three basic activity types that are mapped to the internal transitions, namely *assign*, *empty*, and *terminate*.

The *assign* reads a value from a source data element and writes to another data element. The source may be any term, i.e., a literal value, a variable, or a function call. In this way, the BPEL “opaque” (i.e., non-deterministic) assignment may be modelled using some function with arbitrary interpretation. The activity is represented as an internal transition with the set of corresponding variable assignments.

The *empty* is used to model a “no-op” activity of the process. The
terminate activity can be used to immediately terminate the behavior of a business process instance within which the terminate activity is performed. The activity ends in a special location $s_F$, from which there are no outgoing transitions.

Mapping Structured Activities

The mapping of the structured activities is presented in Table 4.2. The switch activity defines the conditional choice. It consists of an ordered list of branches, defined by case elements, followed optionally by an otherwise branch. The branches are considered in the order in which they appear: the first branch whose condition holds true is taken. If no branch with a condition is taken, then the otherwise branch is taken.

The while activity supports repeated performance of a specified iterative activity. The loop terminates when the condition evaluates to false.

The pick activity awaits the occurrence of one of a set of events and then performs the associated activity. If more than one of the events occurs simultaneously then the choice is nondeterministic. The form of pick is a set of branches of the form event/activity, and exactly one of the branches will be selected based on the occurrence of the associated event. The possible events are the arrival of some message, or an “alarm” based on a timer. We consider the mapping of the alarm event in the following section.

Parallel Execution

In order to specify that the activities are to be executed in parallel, the flow activity is used. The activity completes when all the nested activities complete (or are skipped). The parallel execution is represented as a structure where all the activities are interleaved.

It is possible, however, to express synchronization dependencies that are nested within the flow. The link construct is used for this purposes. It has a
### 4.1. Modelling BPEL Processes

#### Table 4.2: Mapping structured BPEL activities to STS

<table>
<thead>
<tr>
<th>Activity Example</th>
<th>STS representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence</td>
<td>$s_b \xrightarrow{a_1} s_1, s_1 \xrightarrow{a_2} s_2, s_2 \xrightarrow{a_3} s_e$</td>
</tr>
<tr>
<td>activity a1</td>
<td>$s_b, s_1, s_2, s_e \in \mathcal{S}$</td>
</tr>
<tr>
<td>activity a2</td>
<td></td>
</tr>
<tr>
<td>activity a3</td>
<td></td>
</tr>
<tr>
<td>switch</td>
<td>$(s_b, c_1, \tau, \emptyset, \emptyset, true, s_1) \in \mathcal{R}$</td>
</tr>
<tr>
<td>case condition=&quot;c1&quot;</td>
<td>$(s_b, \neg c_1 \land c_2, \tau, \emptyset, \emptyset, true, s_2) \in \mathcal{R}$</td>
</tr>
<tr>
<td>activity a1</td>
<td>$(s_b, \neg c_1 \land \neg c_2, \tau, \emptyset, \emptyset, true, s_3) \in \mathcal{R}$</td>
</tr>
<tr>
<td>case condition=&quot;c2&quot;</td>
<td></td>
</tr>
<tr>
<td>activity a2</td>
<td>$s_1 \xrightarrow{a_1} s_e, s_2 \xrightarrow{a_2} s_e, s_3 \xrightarrow{a_3} s_e$</td>
</tr>
<tr>
<td>otherwise</td>
<td></td>
</tr>
<tr>
<td>activity a3</td>
<td>$s_b, s_1, s_2, s_3, s_e \in \mathcal{S}$</td>
</tr>
<tr>
<td>while</td>
<td>$(s_b, c, \tau, \emptyset, \emptyset, true, s') \in \mathcal{R}$</td>
</tr>
<tr>
<td>condition=&quot;c&quot;</td>
<td>$(s_b, \neg c, \tau, \emptyset, \emptyset, true, s_e) \in \mathcal{R}$</td>
</tr>
<tr>
<td>activity a</td>
<td>$s_b, s', s_e \in \mathcal{S}$</td>
</tr>
<tr>
<td>pick</td>
<td>$s' \xrightarrow{a} s_b$</td>
</tr>
<tr>
<td>onmessage</td>
<td></td>
</tr>
<tr>
<td>variable=&quot;var1&quot;</td>
<td>$(s_b, true, op1(var1), \emptyset, \emptyset, true, s_1) \in \mathcal{R}$</td>
</tr>
<tr>
<td>operation=&quot;op1&quot;</td>
<td>$(s_b, true, op2(var2), \emptyset, \emptyset, true, s_2) \in \mathcal{R}$</td>
</tr>
<tr>
<td>activity a1</td>
<td>$s_1 \xrightarrow{a_1} s_e, s_2 \xrightarrow{a_2} s_e$</td>
</tr>
<tr>
<td>onmessage</td>
<td></td>
</tr>
<tr>
<td>variable=&quot;var2&quot;</td>
<td>$s_b, s_1, s_2, s_e \in \mathcal{S}$</td>
</tr>
<tr>
<td>operation=&quot;op2&quot;</td>
<td>$op1(var1), op2(var2) \in \mathcal{I}$</td>
</tr>
<tr>
<td>activity a2</td>
<td></td>
</tr>
</tbody>
</table>

The source and a target activity, and a link condition. When the source activity completes, the status of the link is set to true if the condition evaluates to true, or false otherwise. The target activity has a *join condition* that depends on the status of all the incoming links. The activity starts only if the join condition evaluates to true. Otherwise the activity is skipped or the exception is fired.
More formally, given a link $l$, let $\phi_l$ be the link condition. By default, $\phi_l = true$. We also define a variable $x_l \in V$ that defines the status of the link. The variable may have one of the following values: \{pos, neg, wait\}. The value of the variable is defined as follows:

- initially $x_l = wait$, the status of the link is undetermined;
- $x_l = pos$ if and only if the source activity is completed and the link condition $\phi_l$ evaluates to true;
- $x_l = neg$ if and only if the source activity is completed and the link condition $\phi_l$ evaluates to false, or the source activity will never be performed (e.g., an alternative branch is selected in a switch, or an alternative event is fired).

The target activity is blocked until all the incoming links are determined: $\forall l, x_l \neq wait$. When this happens, the join condition $\phi_t$ is evaluated. It is defined as a boolean expression over the status variables. If the condition evaluates to true, the activity is fired. If it evaluates to false and the join failure is declared to be suppressed, the activity is skipped. If the condition evaluates to false and the join failure is declared to happen, the corresponding exception $ex$ is fired. The translation is represented in Table 4.3.

**Mapping Timed Activities**

We assume that by default all the BPEL activities are instantaneous, i.e., they are performed immediately when enabled. In this way we neglect the “technical” time and concentrate on the “business” time, i.e, that critical for the business logic. This assumption is formalized using the urgency flag in the corresponding transitions. When for the analysis purposes it is necessary to specify that the activity potentially has a certain duration, it
### 4.1. MODELLING BPEL PROCESSES

Table 4.3: Mapping flow BPEL activity to STS

<table>
<thead>
<tr>
<th>Activity Example</th>
<th>STS representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td>$s_b \xrightarrow{a_1} s_1, s_1 \xrightarrow{a_2} s_e$</td>
</tr>
<tr>
<td>activity a1</td>
<td>$s_b \xrightarrow{a_2} s_2, s_2 \xrightarrow{a_1} s_e$</td>
</tr>
<tr>
<td>activity a2</td>
<td>$s_b, s_1, s_2, s_e, s \in S$</td>
</tr>
</tbody>
</table>

| activity a1      | $(s_b, x_{t1} \neq wait \land x_{t2} \neq wait, \tau, \emptyset, \emptyset, true, s_1) \in \mathcal{R}$ |
| source="l1"     | $(s_1, c, \tau, \emptyset, \emptyset, true, s_2) \in \mathcal{R}$ |
| transitionCondition="c1" | |

| activity a2      | $(s_1, \neg c, \tau, \emptyset, \emptyset, true, s_e) \in \mathcal{R}, \text{ if suppress join failure}$ |
| source="l2"     | $(s_1, \neg c, \tau, \emptyset, \emptyset, true, s_{ex}) \in \mathcal{R}, \text{ if not suppress join failure}$ |
| transitionCondition="c2" | |

| activity a       | $s_2 \xrightarrow{a} s_e$ |
| target="l1"     | $s_b, s_1, s_2, s_e, s_{ex} \in S$ |
| target="l2"     | |
| joinCondition="c"| |

should be expressed explicitly through *duration annotations*. The extended time-related modelling and analysis is discussed in Chapter 7.

Additionally, BPEL defines activities that explicitly refer to time. In particular, the *onAlarm* activity is used to represent timeouts and is modelled as an event handler along with other event handlers. This activity has two variants. In the first variant the event handler is activated when a certain time has passed from entering the activity, e.g., after 5 time units$^2$. This is represented as follows. A dedicated timer $x_t$ is used for the activity. The timer has to be reset when the location, where the activity is defined, is entered. The transition of the activity is modelled as urgent and has a corresponding timed condition. Also a time invariants is assigned to the state, requiring that it should be left no later than the timeout expires.

In the second variant *onAlarm* is fired if the current absolute time has the specified value. This absolute time is modelled with a special timer

---

$^2$The actual syntax of BPEL requires a special syntax to define times in the specification. For the sake of simplicity, we omit their description and refer to the corresponding time units in BPEL.
Another BPEL activity that deals with time is the activity `wait` which blocks the process for a certain time period. The activity also has two forms, where the process waits for a given period or until a given deadline. The activity is represented as a sequence of two transition. The first is urgent, and leads to an intermediate location, where the corresponding location invariant and urgent transition are defined. The mapping of the timed constructs is represented in Table 4.4.

<table>
<thead>
<tr>
<th>Activity Example</th>
<th>STS representation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pick</code></td>
<td></td>
</tr>
<tr>
<td>onmessage</td>
<td></td>
</tr>
</tbody>
</table>
| variable="v" operation="op" activity a1 | \((s_b, \text{true, } \overline{\text{op}}(v), \emptyset, \emptyset, \text{true}, s_1) \in \mathcal{R}
\) \((s_b, x_t = 5, \tau, \emptyset, \emptyset, \text{true}, s_2) \in \mathcal{R}
\)
|                  | \(\forall (s', \phi, a, \Omega, \vartheta, s_b) \in \mathcal{R}, Y = \{x_t\}
\) \(L_i(s_b) \equiv x_t \leq 5
\)
|                  | \(s_b, s_1, s_2, s_e \in \mathcal{S}, s_1 \xrightarrow{a_1} s_e, s_2 \xrightarrow{a_2} s_e
\) |
| onalarm          |                    |
| for="5"         |                    |
| activity a2      |                    |
|                  | \((s_b, \text{true, } \overline{\text{op}}(v), \emptyset, \emptyset, \text{true}, s_1) \in \mathcal{R}
\) \((s_b, \text{global} = 5, \tau, \emptyset, \emptyset, \text{true}, s_2) \in \mathcal{R}
\)
|                  | \(L_i(s_b) \equiv \text{global} \leq 5
\)
|                  | \(s_b, s_1, s_2, s_e \in \mathcal{S}, s_1 \xrightarrow{a_1} s_e, s_2 \xrightarrow{a_2} s_e
\) |
| wait             |                    |
| for="5"         |                    |
|                  | \((s_b, \text{true, } \tau, \emptyset, \{x_t\}, \text{true}, s') \in \mathcal{R}
\) \((s', x_t = 5, \tau, \emptyset, \emptyset, \text{true}, s_e) \in \mathcal{R}
\)
|                  | \(L_i(s') \equiv x_t \leq 5, s_b, s', s_e \in \mathcal{S}
\) |
| wait             |                    |
| until="5"       |                    |
|                  | \((s_b, \text{true, } \tau, \emptyset, \emptyset, \text{true}, s') \in \mathcal{R}
\) \((s', \text{global} = 5, \tau, \emptyset, \emptyset, \text{true}, s_e) \in \mathcal{R}
\)
|                  | \(L_i(s') \equiv \text{global} \leq 5, s_b, s', s_e \in \mathcal{S}
\) |
### Table 4.5: Mapping fault and compensation BPEL activities to STS

<table>
<thead>
<tr>
<th>Activity Example</th>
<th>STS representation</th>
</tr>
</thead>
</table>
| throw faultName="f" faultVariable="fv" | \((s_b, true, \tau, \emptyset, \emptyset, true, s_f) \in \mathcal{R}\)  
| |  
| | \(s_b, s_f \in \mathcal{S}\)  
| catch faultName="f" faultVariable="fv" activity a | \(s_f \xrightarrow{a} s_e\)  
| | \(s_f, s_e \in \mathcal{S}\)  
| compensate scope="sc" | \((s_b, true, \tau, \emptyset, \emptyset, true, s_{csc}) \in \mathcal{R}\)  
| |  
| | \(s_b, s_{csc} \in \mathcal{S}\)  
| scope name="sc" compensationHandler activity a | \(s_{csc} \xrightarrow{a} s_e\)  
| | \(s_{csc}, s_e \in \mathcal{S}\)  

#### Fault and Compensation Handlers

The translation of fault- and compensation-related activities is presented in Fig. 4.5. In BPEL a fault may be raised on the receipt of a message, when a synchronous invoke operation waits for reply. Alternatively, the fault may be defined to be thrown using BPEL `throw` activity. A fault handler (catch declaration) itself is a simply a wrapper for a specific activity to be performed. Note that the normal execution is interrupted on a fault and the control passes to the fault handling activity. A fault also completes the current unit of work (BPEL scope).

A compensation handler is always associated with a work unit (BPEL scope), and is invoked (explicitly or implicitly) using the BPEL `compensate` activity. The name of the compensated work unit (scope) may be given as input to the compensation in order to define a specific ordering for the compensation of enclosed activities. The full semantics of BPEL assumes, however, a more complex model. When the scope instance completes successfully, the corresponding compensation handler instance is installed. First, this instance takes a snapshot of the current values of the variables, and uses it when the compensation is activated. Second, an arbi-
trary number of such scopes can be created, e.g., when the corresponding scope is defined within the loop. When compensated, all the currently installed instances should be activated. More detailed analysis of the full semantics and its modelling is out of the scope of this work.

**Restrictions of the Translation**

The mapping from BPEL to STS is based on several simplification assumptions. These assumptions are dictated by the necessity to carry out the analysis tasks, that are not feasible (if at all possible) otherwise. While these assumptions do not permit the analysis of an arbitrary composition of BPEL processes, they allow to formalize and reason on a wide range of important classes of problems and scenarios that are covered only partially by many of the existing approaches and models.

In our model we assume that the composition is fixed. That is, the number of active processes is fixed and the processes can not be created or destroyed dynamically. First, this means a fixed number of participating service instances. This puts additional restrictions on the usage of dynamic binding of partners of the process through the a mechanism of *endpoint references*, and the aligning of messages to process instances using *correlation* in BPEL. The messages are related with a particular process instance using so-called *correlation sets*, that is special information tokens uniquely identifying the relation. This mechanism plays an important role in the compositions with an arbitrary number of instances and dynamic instance creation. The presented formalism allows us to reflect these mechanisms using a predefined number of instances, a fixed set of endpoint references and correlation tokens.

Second, the number of instantiated subprocesses should also be fixed. In BPEL the subprocesses are dynamically created in two cases: when a compensation handler instance is installed, and when a message event
4.2 MODELLING COMPOSITION REQUIREMENTS

handler (a structured activity associated to a scope, activated on a message reception, and executed in parallel with the main process) is dispatched concurrently. Such a behavior creates an additional source of analysis complexity and is currently omitted.

The semantics of the management of variables and time constraints is also simplified. In particular, the set of process variables should be finite, the XML data structures should also have finite representation, etc. See [57] for more details. For what concerns time, the time conditions are restricted to comparison of timers with the constant values. On the contrary, BPEL allows for more complex constructs. This restriction is dictated by the decidability problems of the timed analysis, as discussed in the following sections.

In the future, we are going to relax these omissions, addressing a wider class of problems. We also looking toward implementation of other existing BPEL constructs (e.g., serialization of scopes, predefined faults, etc.), as well as the new constructs to be presented in the future versions of the standard.

4.2 Modelling Composition Requirements

The goal of the formal analysis of the Web service compositions is to verify the correctness of the composition specification with respect to a set of behavioral requirements. These requirements represent various properties on the global and local evolution of the system, data modifications, ordering constraints between events, etc. We see the following ways to exploit them in the composition modelling:

- to define the required properties of the composite system (assertions). Such requirements express the global or local policies and rules that should be satisfied in every execution of the application, regardless a
particular business scenario applied. Deadlock freeness, consistency of data or transactional context are the examples of assertion properties.

- to define the desired properties of the composite system (possibilities). Requirements of this type express the goals of the composite application, e.g., the possibility to successfully complete the purchase order transaction, the ability to fulfill an information or travel booking request. The possibility property is expected to be satisfied in some execution scenario of the composite system.

- to define the assumed properties of the Web service composition. These properties express certain facts that the designer expects to be enforced by some participant, thus modelling its obligations. In the analysis process such properties are used to constrain the model of the composition. An actual service that will implement a corresponding role in the composition is responsible to satisfy these requirements at run-time (e.g., pre- and post-conditions over service operations).

In order to represent various behavioral requirements we exploit temporal logic formalism with linear model of time. Although other variants of time interpretations are used in literature, the linear model allows for more natural and adequate representation of behavioral properties [134]. While we model properties as logic formulas, a visual characterization may be used for these purposes (e.g., as Message Sequence Charts or property patterns), provided that analogous semantics is given.

4.2.1 Linear-time Temporal Logic

Linear-time Temporal Logic (LTL for short, [48]) is a formal language for specifying and reasoning about the evolution of the underlying system over time. As it follows from its name, the logic adopts the linear model
of time, that is the behavior of the system is modelled as a set of infinite executions\(^3\).

We use LTL to express properties over the behavior of the Web service compositions. In particular, we are interested in the evolution of the states of the processes or a composition as a whole (control flow properties), in the evolution of interactions among the participants (message flow properties), and in the evolution of data objects (data flow properties). Therefore, the specified properties may be defined over the process locations, over the communication actions, and over the process variables. Note that the queue variables and structures are not used for the specifications of the properties, since the communication model is implementation-dependent.

We remark also that the analysis of time flow properties of the composition require more advanced features than those provided by LTL, and is discussed separately in Chapter 7.

More formally, the logic is defined as follows. We use temporal operators □ (“always”), ◯ (“eventually”), U (“until”), R (“release”). We do not consider “next” operator, since its semantic makes few sense for the execution of asynchronous systems [71]. We also use a predicate ACTION(a) to denote that the current action performed is \(a\), and INSTATE(s, i) to denote that the current location of the \(i^{th}\) process is \(s\).

- \(t_1 = t_2\) is a formula if for \(i = 1, 2\), \(t_i \in V^j\), or \(t_i \equiv f(x_1, \ldots, x_n), f \in F\);
- ACTION(a) is a formula if \(a \in A\);
- INSTATE(i, s) is a formula if \(s \in S^i\);
- if \(\phi_1, \phi_2\) are the formulas, then \(\neg \phi_1, \phi_1 \land \phi_2, \phi_1 \lor \phi_2\) are the formulas;
- if \(\phi_1, \phi_2\) are the formulas, then \(\Diamond \phi, \Box \phi, \phi_1 U \phi_2, \phi_1 R \phi_2\) are the formulas.

\(^3\)We remark that while in our formalism the execution may be finite terminating in some final location, the infinite execution may be emulated by adding a fake self loop to such a final location.
Given a run \( \omega = \sigma_0, a_0, \sigma_1, a_1, \ldots \) of the composition behavior \( \Xi \), and a LTL formula \( \phi \), we write \( \omega \models \phi \) to denote that the formula \( \phi \) holds on the run \( \omega \). We also write \( \omega^i \) to denote the suffix of \( \omega \) starting at \( \sigma^i \). The relation \( \models \) is defined as follows:

\[
\begin{align*}
\omega \models (t_1 = t_2) & \iff \sigma_0 = (\bar{s}, \bar{g}, C, \bar{u}) \text{ and } \bar{g} \models (t_1 = t_2). \\
\omega \models \text{ACTION}(a) & \iff a_0 = a. \\
\omega \models \text{INSTATE}(i, s) & \iff \bar{s}(i) = s. \\
\omega \models \neg \phi & \iff \omega \not\models \phi. \\
\omega \models \phi_1 \lor \phi_2 & \iff \omega \models \phi_1 \text{ or } \omega \models \phi_2. \\
\omega \models \phi_1 \land \phi_2 & \iff \omega \models \phi_1 \text{ and } \omega \models \phi_2. \\
\omega \models \Diamond \phi & \iff \text{for some } i \geq 0, \omega^i \models \phi. \\
\omega \models \Box \phi & \iff \text{for every } i \geq 0, \omega^i \models \phi. \\
\omega \models \phi_1 \mathcal{U} \phi_2 & \iff \text{there exists } i \geq 0, \text{ such that } \omega^i \models \phi_2, \text{ and for every } 0 \leq j < i, \omega^j \not\models \phi_1. \\
\omega \models \phi_1 \mathcal{R} \phi_2 & \iff \text{for all } i \geq 0, \text{ if for every } 0 \leq j < i, \omega^j \not\models \phi_1, \text{ then } \omega^i \models \phi_2.
\end{align*}
\]

It is easy to see that the following relation between the operators holds:

\[
\begin{align*}
\bullet \ \phi_1 \land \phi_2 & \equiv \neg (\neg \phi_1 \lor \neg \phi_2) \\
\bullet \ \Diamond \phi & \equiv \text{true} \mathcal{U} \phi \\
\bullet \ \Box \phi & \equiv \neg (\Diamond \neg \phi) \\
\bullet \ \phi_1 \mathcal{R} \phi_2 & \equiv \neg (\phi_1 \mathcal{U} \neg \phi_2)
\end{align*}
\]

### 4.2.2 Interpretation of the Property Categories

The meaning of an LTL formula depends on the usage of the corresponding requirement in the Web service composition model. We distinguish constraints, assertion properties, and possibility properties.
The possibility properties describe the scenarios that are desired by the composition designers, i.e., an ability to reach a positive payment acknowledgement resulted from a business transaction, an ability to perform certain operation, etc. Intuitively, the requirement is that there should exist an execution of the composition where the property holds.

**Definition 4.1 (Possibility Satisfiability)**

$\Sigma_\Delta$ satisfies a possibility $\phi$, written as $\Sigma_\Delta \models_P \phi$, iff there exists a run $\omega \in \Xi(\Sigma_\Delta)$, s.t. $\omega \models \phi$.

The assertions express the properties that are expected to hold regardless a particular execution scenario. Deadlock and livelock freeness are the examples of the assertion properties. In other words, the assertions are expected to hold in all the scenarios for the given composition specification.

**Definition 4.2 (Assertion Satisfiability)**

$\Sigma_\Delta$ satisfies an assertion $\phi$, written as $\Sigma_\Delta \models_A \phi$, iff for all runs $\omega \in \Xi(\Sigma_\Delta)$, $\omega \models \phi$.

The constraints are used to model assumptions, that is, the properties that are not verified, but are considered to be true in all the execution of a system.

**Definition 4.3 (Constrained Satisfiability)**

$\Sigma_\Delta$ satisfies a possibility $\phi$ under a constraint $\psi$, written as $\Sigma_\Delta, \psi \models_P \phi$, iff there exists a run $\omega \in \Xi(\Sigma_\Delta)$ such that $\omega \models \phi \land \psi$.

$\Sigma_\Delta$ satisfies an assertion $\phi$ under a constraint $\psi$, written as $\Sigma_\Delta, \psi \models_A \phi$, iff for each run $\omega \in \Xi(\Sigma_\Delta)$, if $\omega \models \psi$ then $\omega \models \phi$.

**Example 4.3** Consider the Flight Reservation process presented in Fig. 4.1. An assertion that the process terminates only in one of its final states (i.e., “NA”, “FAIL”, or “DONE”) may be expressed with the following formula:

$\Diamond (\text{INSTATE(Flight, na)} \lor \text{INSTATE(Flight, fail)} \lor \text{INSTATE(Flight, done)})$
Analogously, the possibility to eventually emit the flight tickets in reply to the acknowledgement from the user is described as

\[ \Diamond \text{ACTION}(\text{reply-ack}) \].

### 4.3 Discussion

In this chapter we demonstrated how the formal model presented in Chapter 3 may be applied to BPEL and WSDL, the standard languages for the specification of Web service compositions. Providing a mapping between the STS model and a BPEL/WSDL specification is not a main focus of this thesis, and therefore the presented translation is rather high-level and informal. However, this translation procedure allows one to easily represent a wide class of BPEL constructs and specifications, and is used in our framework as a basis for the formal analysis of Web service composition specifications.

There were many research efforts towards formalizing (the semantics of) BPEL using various models, such as Petri Nets, process algebra, finite state automata, etc (see [130] for the detailed overview). As well as the mapping presented here, these works usually provide only partial coverage of the whole language. We remark, however, that the integrated support of the data- and time-related constructs in our formalism allows in many cases for a more complete representation of the BPEL specifications.

We also introduced a formal language that allows to describe behavioral and functional requirements, such as deadlocks, event ordering constraints, data evolution, etc. The use of linear time semantics allows for more natural and intuitive representation of behavioral properties. Analogous formalisations are exploited in some other works on the analysis of the Web service compositions, e.g. [101, 53, 56]. Visual notations, as those presented in [133], may be exploited for the definition of such properties.
4.3. DISCUSSION
Chapter 5

Analysis of Communication Models

In the analysis of distributed applications the model adopted for representing the communications between components plays an important role. Indeed, these communications are usually asynchronous and buffered, message overpasses (i.e., reorderings) are possible, hence an accurate representation of message passing protocol is needed in order to ensure application correctness. In the Web services domain this problem becomes even more critical. Since different platforms may be used in the same distributed application, and, more important, the information about their implementations is often abstracted away, it is not possible to apply some fixed communication model for the analysis of the composition behavior.

Here we present an analysis approach, based on a parametric model, that allows us to capture a wide hierarchy of communication systems. Using a set of examples, we show that different scenarios require different communication models, and that the parametric model suites well for their description. The approach allows us to associate with a composition the most adequate communication model, i.e., the simplest model that is sufficient to capture all the behaviors of the composition. In this way, we can also achieve higher efficiency in verification, and deduce a minimal set of requirements to the underlying system that guarantee the correctness.
5.1 Case Study: Virtual Travel Agency

In order to illustrate the problem of modeling service compositions, we consider several variants of the Virtual Travel Agency domain. The goal of the Virtual Travel Agency is to provide a combined flight and hotel booking service by integrating two independent existing services: a Flight booking service, and a Hotel booking service. Thus, the composition describes the interactions of four partners: User, Virtual Travel Agency (VTA), Hotel and Flight services (see Fig. 5.1.a).

5.1.1 Example 1: Tickets Reservation Scenario

In this scenario, the user can ask the VTA, to book a flight to a specified location and reserve a room in a hotel at that location for a given period of time. It is possible that the request of the user cannot be fulfilled, in which case the user receives a not-available (na) notification from the VTA. If a reservation offer is received instead, the user can accept or reject it, sending a corresponding message to the VTA (Fig. 5.1.b).

The Flight booking service becomes active upon a request for a given location (e.g., Paris) and a given period of time (e.g., August). In the case the booking is not possible, this is signaled to the requestor, and the protocol terminates. Otherwise, the requestor is notified with an offer information and the protocol stops waiting for either a positive or negative acknowledgment. In case of positive answer, the flight is successfully booked and the reservation ticket is sent, otherwise the interaction terminates with failure. Figure 5.1.c represents the protocol provided by the Flight booking service. The protocol of the Hotel service is similar.

The behavior of the VTA is as follows. After receiving a reservation request from the user, the VTA interacts with Flight and Hotel services to obtain ticket offers and expects either a negative answer if this is not
possible (in which case the user is notified and the protocol terminates with failure), or provides the user with an offer indicating hotel, flights and cost of the trip. After that, the user may either accept or refuse the offer, and in the first case the VTA provides the user with the tickets obtained from Hotel and Flight. The diagram corresponding to the BPEL protocol of the VTA is represented in Fig. 5.1.d.
This composition scenario exhibits an important property that allows for a very simple communication mechanism. At any moment of time, only one of the partners is ready to emit a message. Moreover, the corresponding receiver is ready to accept the message. Using the terminology of [56], the composition model satisfies the synchronous compatibility, autonomy and lossless composition properties. As a consequence, a model, where the partners synchronize on shared actions (and, therefore, the queues are not needed), may be used without affecting the correctness of the verification. As demonstrated in [56], this allows for an much more efficient verification of the composition scenarios.

5.1.2 Example 2: Reservation with Cancellation

Unfortunately, the simplified communication model of the previous example is not applicable to all kinds of interactions. A typical example is a business process with event handlers. Let us consider an extension of the ticket reservation scenario, such that the user can decide to cancel the booking operation. In this case the user can send a cancel message to the VTA and wait for the outcome of the cancellation. The VTA forwards the cancellation to the Flight (and similarly to the Hotel process, we omit this for the sake of simplicity). The Flight waits for a cancellation message for a certain time after the acknowledgement of the reservation. If the cancellation message is received on time, the Flight notifies a successful cancellation. If the time for a cancellation runs out, the Flight sends a ticket to the VTA, thus forcing the failure of cancellation: cancellations sent by the VTA after the ticket is sent are consumed and ignored. The excerpts of the corresponding process specifications are represented in Fig. 5.2.

The verification under the synchronous communication model is not able to manage correctly this example, and reports a deadlock. Indeed, if the Flight service fails to wait for a cancellation, the onAlarm activity
is fired and a ticket is sent to the VTA process. Meanwhile, the VTA may receive a cancellation message from user and forward it to the Flight service. Therefore both Flight and VTA would try to send messages to each other and the composition would be in a deadlock according to the synchronous semantics.

This deadlock is not real, in the sense it does not occur in existing BPEL engines; since the Web services communications are asynchronous, and the message emission is not blocking, both processes will emit messages to each other. Both messages will be consumed then and the composition terminates correctly.

The problem we are facing here is that the synchronous model is too restrictive. The message delivery and processing may require a certain time, leading to situations where concurrent emissions take place. These situations, however, are not allowed in the synchronous communication model. In order to verify correctly the considered example, a more general model is needed that allows for modelling these concurrent message emissions.

We remark that if one applies the verification approach of [56] for the analysis of such a composition, then the inapplicability of the synchronous communication model is detected and reported. However, [56] and other
verification approaches fail to find an alternative communication model that is adequate to the scenario. In [56], for instance, a communication model with 2-position buffers for each participant is applied, which may be rather expensive for the verification, and still may not guarantee in general the correctness of the verification. In case of the scenario in Fig. 5.2 this approach would lead to the correct results, while this is not the case for the scenario we present below.

### 5.1.3 Example 3: Extended Cancellation

Let us consider a further modification of the case study. In this case, after the VTA has sent a cancellation message to the Flight, it waits for a message notifying whether the cancellation is possible (message `flight cancelled`) or not (message `no flight cancel`). In the latter case, it waits for the ticket and sends it to the user. The Flight service, on the other side, behaves as before with the only difference that, after emitting the ticket and receiving the cancellation, it sends a notification about cancellation rejection (message `no flight cancel`). The corresponding diagrams are represented in Fig. 5.3.

---

Figure 5.3: Virtual Travel Agency composition: Example 3
Even if one verifies the example allowing for concurrent message emissions, the following incorrect scenario may occur. The Flight service sends a ticket and waits for a cancellation. At the same time, the VTA process sends a cancellation request that the Flight service rejects. The VTA has received a ticket and then a cancellation rejection, but it is not able to process the messages in this order. Only if the execution of processes in the run-time environment allows for reordering of messages (which is the case for existing implementations) the deadlock disappears, since the cancellation rejection can be processed before the ticket message.

This example shows a necessity not only to consider systems which do not follow the synchronous communication semantics, but also to accept less restrictive models where message reordering is allowed. If this is not done, then scenarios that can occur in practice are not considered in the verification, and wrong results can be obtained.

5.2 Problem Formalization

The chain of the composition scenarios can be further prolonged, leading to more and more sophisticated communication models. Different models may entail different behaviors in the same composition scenario, and, therefore, the result of the verification of a composition depends on the selected communication model.

The problem that one has to face in these settings is twofold. First, it is needed to be able to uniformly describe different communication models and there relations with respect to the given composition scenario. For this purpose we use the parametric communication model defined in Section 3.3.2. We show how this parametric model allows us to represent various communication patterns (including those discussed in Examples 1, 2, 3), and how various communication model form a hierarchy, which we
will use in our analysis.

Second, it is needed to identify the communication model that is appropriate for the given scenario, i.e., is able to fully describe the behavior of the composition. The algorithm we present in Section 5.3 aims at addressing this problem.

Additionally to the composition assumptions discussed in Section 3.1, we require a correct system to satisfy the following requirements.

We consider systems with channels that grow unboundedly as “bad” systems. The analysis should be able to identify and rule out such systems. We say that the channel of a composition have a *bounded growth* if, for every queue $q_i$, either a finite bound $b_i < \infty$ is declared, or there is some constant $K_i$ such that the queue contains at most $K_i$ messages in all reachable states. We say that the Global Transition System (GTS, Section 3.3.3) is *bounded* if it has finite number of queues and bounded growth.

We also require the composition to be *complete*. We say that the GTS is *complete* if all the terminating configurations $\langle \bar{s}, \bar{g}, C, \bar{u} \rangle$ (that is, leaves of the reachability tree, or the nodes with only outgoing time passing transitions) have empty queue content: $C = \langle \epsilon, \ldots, \epsilon \rangle$. The systems that are not complete loose messages: indeed, at the end of the computation there are unconsumed messages in queues. We will consider only complete GTS in the following.

## 5.3 Hierarchy of Models

In this section we address the problem of defining suitable communication models for the composition, and of guaranteeing that these communication models are *adequate* w.r.t. the real executions, i.e., that they do not discard any execution that can happen according to our assumptions.

This is achieved through the following steps:
• We define the “most general” model in terms of GTS. This is a model which allows more behaviors than any other does.

• We define a family of possible communication models that we can adopt for the verification of composite systems. These communication models correspond to different levels of complexity and efficiency of the verification process. All the models we propose are expressible in terms of GTS, by changing the queue model.

• We define the “adequacy” of a communication model for a composite system: a communication model is adequate if it expresses all the behaviors of the most general model, i.e., no behaviors are lost due to the specific queuing model.

• For each communication model, we also discuss the requirements on the middleware that guarantee that all the behaviors expressed by the model can happen in real implementations of BPEL engines.

5.3.1 Relations among Communication Models

One of the tasks in the adequacy analysis is to check whether the composition of Web services under the given communication model does not loose behaviors w.r.t. some other more general model. This requires introduction of certain relations between models, namely simulation relations.

**Definition 5.1** We say that a configuration \( \gamma_2 \) of \( \Sigma_{\Delta_2} \) simulates a configuration \( \gamma_1 \) of \( \Sigma_{\Delta_1} \), written as \( \gamma_1 \preceq \gamma_2 \), iff

- \( \sigma(\gamma_1) = \sigma(\gamma_2) \),

- for any \( a \), for any \( \gamma_1' \), if \( \gamma_1 \xrightarrow{a} \gamma_1' \), then there exists \( \gamma_2' \), such that \( \gamma_2 \xrightarrow{a} \gamma_2' \), and \( \gamma_1' \preceq \gamma_2' \).
We say that $\Sigma_\Delta_2$ simulates $\Sigma_\Delta_1$, written as $\Sigma_\Delta_1 \preceq \Sigma_\Delta_2$, iff for any $\gamma_0 \in \Gamma_0$ there exists $\gamma_2 \in \Gamma_2$ such that $\gamma_0 \preceq \gamma_2$.

We say that $\Sigma_\Delta_1$ and $\Sigma_\Delta_2$ are equivalent, written as $\Sigma_\Delta_1 \approx \Sigma_\Delta_2$, iff $\Sigma_\Delta_1 \preceq \Sigma_\Delta_2$ $\land$ $\Sigma_\Delta_2 \preceq \Sigma_\Delta_1$.

**Proposition 5.1**

If $\Sigma_\Delta_1 \preceq \Sigma_\Delta_2$, then $\Xi(\Sigma_\Delta_1) \subseteq \Xi(\Sigma_\Delta_2)$.

If $\Sigma_\Delta_1 \approx \Sigma_\Delta_2$, then $\Xi(\Sigma_\Delta_1) = \Xi(\Sigma_\Delta_2)$.

**Proof.** Consider a run $\omega = \sigma_0, a_0, \sigma_1, a_1, \cdots \in \Xi(\Sigma_\Delta_1)$. It is easy to see that this run also belongs to $\Xi(\Sigma_\Delta_2)$. Indeed, by definition of behavior, there exists a sequence $\pi_1 = \gamma_0, a_0, \gamma_1, a_1, \cdots$, such that $\gamma_0 \xrightarrow{\pi_1} ^*$, and $\sigma_i = \sigma(\gamma_i)$. From $\Sigma_\Delta_1 \preceq \Sigma_\Delta_2$ follows that there exists $\pi_2 = \gamma_0, a_0, \gamma_1, a_1, \cdots$, such that $\gamma_0 \xrightarrow{\pi_2} ^*$, and for each $i \geq 0$, $\gamma_i \preceq \gamma_i$. Since $\gamma_i \preceq \gamma_i$ implies that $\sigma(\gamma_i) = \sigma(\gamma_i)$, then $\omega$ is also fireable in $\Sigma_\Delta_2$, and therefore $\omega \in \Xi(\Sigma_\Delta_2)$. The second part can be proved analogously. □

When a simulation relation between two communication models $\Delta_1$ and $\Delta_2$ holds for any set of STSs, we say that the model $\Delta_2$ is *more general* than $\Delta_1$.

**Definition 5.2** Communication model $\Delta_2$ simulates model $\Delta_1$, written as $\Delta_1 \sqsubseteq \Delta_2$, if for every composition $\Sigma$ of STSs, $\Sigma_\Delta_1 \preceq \Sigma_\Delta_2$.

Being reflexive and transitive, this relation forms a partial order on the set of communication models. Below we identify a “most general” model, that is a model $\Delta_{MG}$, such that for any other model $\Delta$ holds $\Delta \sqsubseteq \Delta_{MG}$.

The simulation relation relies on the structure of the queues. There are two dimensions in which models differ. First, it depends on the queue bounds: the bigger a queue bound is, the more transitions are enabled. Second, it depends on the distribution of the alphabets: if the alphabet of
each ordered queue in one model is a subset of the alphabet of some ordered queue in another, then the first model is more general then the other. The following theorem defines relation between models with different queue structures.

**Theorem 5.1** Consider two communication models $\Delta_1 = \langle B_1, L_{1M}, L_{1O} \rangle$ and $\Delta_2 = \langle B_2, L_{2M}, L_{2O} \rangle$. If for each queue $q_{2i}$ of $\Delta_2$ holds that

- if the queue $q_{2i}$ is ordered, then there exists an ordered queue $q_{1j}$, such that $M_{2i} \subseteq M_{1j}$, and
- $b_{2i} \geq \sum_{M_{2i} \cap M_{1j} \neq \emptyset} b_{1j}$,

then $\Delta_1 \sqsubseteq \Delta_2$.

**Proof.** We will consider an arbitrary composition $\Sigma$ under both models. Consider an arbitrary configuration $\gamma_1 = \langle \bar{s}_1, \bar{g}_1, C_1, \bar{u}_1 \rangle$ of $\Sigma_{\Delta_1}$. We construct the configuration $\gamma_2 = \langle \bar{s}_1, \bar{g}_1, C_2, \bar{u}_1 \rangle$ of $\Sigma_{\Delta_2}$ as follows. Let us denote the number of messages $\alpha$ in a queue content $w$ as $|w|_\alpha$. If the queue $q_{2i}$ is ordered, then $w_{2i}$ is a projection of the string $w_{1j}$ on $M_{2i}$. If the queue $q_{2i}$ is unordered, then $w_{2i}(\alpha) = |w_{1j}|_\alpha$, with $\alpha \in M_{1j}$. When this holds, we write $C_2 = C_1 \downarrow$. Due to the premise of the theorem and the fact that the queue alphabets are disjoint, this construction is possible and unique. We will denote the relation induced by this construction as $\gamma_1 \Rightarrow \gamma_2$ and show that it is a simulation relation.

Consider an arbitrary action $a$ fireable in $\gamma_1$: $\gamma_1 \xrightarrow{a} \gamma_1'$, where $\gamma_1' = \langle \bar{s}_1', \bar{g}_1', C_1', \bar{u}_1' \rangle$.

- if $a = \tau$ then $C_1' = C_1$, and $\gamma_2 \xrightarrow{a} \gamma_2'$, where $\gamma_2' = \langle \bar{s}_1', \bar{g}_1', C_2, \bar{u}_1' \rangle$. Therefore, $\gamma_1' \Rightarrow \gamma_2'$.

- if $a = \mu$ then $C_1' = C_1.\alpha$, $\text{OP}(\alpha) = \mu$. Therefore, $w_{1j}' = w_{1j}.\alpha$, if $\alpha \in M_{1j}$, and $w_{1j}' = w_{1j}$ otherwise. Consider $\gamma_2' = \langle \bar{s}_1', \bar{g}_1', C_2', \bar{u}_1' \rangle$,
with \( C'_2 = \langle w'_21, \ldots, w'_2l \rangle \) such that \( w'_2i = w_{2i}.\alpha \), if \( \alpha \in M_{2i} \), and \( w'_2i = w_{2i} \) otherwise. It is easy to see that \( C'_2 = C'_1 \downarrow \). Therefore, for any queue \( q_{2i} \), \( |w'_2i| = \sum_{\alpha \in M_{2i}} |w'_{2i}| \alpha = \sum_{\alpha \in M_{2i} \cap \alpha \in M_{1j}} |w'_{1j}| \alpha \leq \sum_{M_{2i} \cap M_{1j} \neq \emptyset} b_{1j} \leq b_{2i} \). As a result, \( \gamma'_1 \Rightarrow \gamma'_2 \), and \( \gamma_2 \xrightarrow{a} \gamma'_2 \).

- \( a = \overrightarrow{\mu} \): then \( C_1 = \alpha.C'_1, \text{OP}(\alpha) = \mu \). That is, there exists a queue \( q_{1j} \) s.t. \( w_{1j} = \alpha.w'_{1j} \). Let us define \( w'_2i = w_{2i} \), if \( \alpha \notin M_{2i} \). If \( \alpha \in M_{2i} \) and \( q_{2i} \) is ordered, then also \( q_{1j} \) is. Therefore, from \( C'_2 = C'_1 \downarrow \) follows that there exists \( w'_{2i} \), such that \( w_{2i} = \alpha.w'_{2i} \). If the queue is unordered, let us define \( w'_2i \) the same as \( w_{2i} \) except \( w'_{2i}(\alpha) = w_{2i}(\alpha) - 1 \). It follows that \( C'_2 = C'_1 \downarrow, \text{action is fireable in } \gamma_2, \text{and may lead to a configuration } \gamma'_2 = \langle \bar{s}'_1, \bar{g}'_1, C'_2, \bar{u}'_1 \rangle, \text{such that } \gamma'_1 \Rightarrow \gamma'_2 \).

Consequently, \( \gamma_1 \) and \( \gamma_2 \) have the same observable state; each transition enabled in \( \gamma_1 \) is also enabled in \( \gamma_2 \) leading to a state that preserves the relation. As a result, \( \Rightarrow \) is a simulation relation: \( \gamma_1 \preceq \gamma_2 \). Since the choice of the configuration and composition was arbitrary, it follows that \( \Delta_1 \sqsubseteq \Delta_2 \). □

The theorem may be easily understood on the following example. Consider a model \( \Delta_1 \) with one ordered queue \( q_1 \) with alphabet \( M = \{\alpha_1, \alpha_2\} \), and a model \( \Delta_2 \) with two queues \( q_{21} \) and \( q_{22} \) with alphabets \( \{\alpha_1\} \) and \( \{\alpha_2\} \) respectively. If an input action is allowed in the composition under model \( \Delta_1 \) then it is also allowed in the second model, since if a message is on the top of the queue in first model and can hence be consumed, then it is on the top of one of the queues in the second model. Similarly, if an output action \( \overleftarrow{\mu}_1 \) is allowed in the first model, then the queue is not full, that is \( |q_1| < b_1 \). Since \( |q_{21}| \leq |q_1| \) (the two queues have the same length if \( q_1 \) contains only \( \alpha_1 \) messages) and, by hypothesis \( b_1 \leq b_{21} \), then \( |q_{21}| < b_{21} \) and hence the output action \( \overleftarrow{\mu}_1 \) is not blocked in the second model.
5.3.2 Most General Communication Model

The first step of the adequacy analysis is to define the reference model, that is the model that allows for the largest set of behaviors. In order to respect the assumptions presented above, this model has to allow for potentially unbounded queues, non-blocking emissions, and arbitrary, unordered access to the content of any queue.

**Definition 5.3** The Most General Communication Model (MG-model) is a communication model \( \Delta_{MG} = \langle B, L_M, L_O \rangle \), with 1 unordered queue, \( b = \infty \), and \( L_M(\alpha_i) = 1 \).

It is easy to see that such a model is indeed a generalization of any other communication model w.r.t. the behavior of any composition of STSs.

**Proposition 5.2** For any communication model \( \Delta \), \( \Delta \sqsubseteq \Delta_{MG} \).

**Proof.** Follows immediately from Theorem 5.1. \( \square \)

Whenever a composition under a certain model \( \Delta \) simulates the most general composition, we say that this model is *adequate* for the description of the composition scenario.

**Definition 5.4** A communication model \( \Delta \) is said to be adequate for the given composition scenario if \( \Sigma_\Delta \approx \Sigma_{\Delta_{MG}} \).

**Implementation requirements.** Model \( \Delta_{MG} \) defines the most liberal policy for the message handling: each message can be accessed and consumed regardless the reception order. On the other hand, this model is also the least realistic, among the ones described in this section, for what concerns the implementation of a middleware generating all the behaviors allowed by the model. Indeed, all existing engines apply a specific policy for the queues and do not allow for such an arbitrary consumption of messages.
5.3. HIERARCHY OF MODELS

5.3.3 Interpretation of Communication Models

We now define a hierarchy of communication models that are particularly significant for verifying Web service compositions and that have been proposed in the literature.

Synchronizable Communications.

This is the most restricted communication model that can be defined in terms of GTS formalization. In this model there is only one queue of capacity one.

Definition 5.5 The synchronizable communication model is the model \( \Delta_1 = \langle B, L_M, L_O \rangle \), with \( B = \{1\} \) and \( L_M(\alpha) = 1 \) for all messages \( \alpha \).

This model is strongly related to another communication model widely used for modeling Web service compositions, namely synchronous composition. In such a model, communicating processes synchronize on shared actions; therefore this model can be represented without queues. More precisely, when the \( \Delta_1 \) model is shown to be adequate for a given composition scenario, and the composition is complete, one can use a synchronous composition for the analysis of wide range of properties, thus achieving better performance. In the following, we formalize this result.

We start with a definition of a synchronous composition. In such a composition the partners synchronize on a shared communication operation, i.e., execute it simultaneously. We identify a shared action by the corresponding message type \( \mu \). Internal actions are executed independently. The key feature of the synchronous composition is that the queues are not needed, thus simplifying the representation and analysis of the composition. A set of all actions in the composition is denoted as \( \mathcal{A} = \{\tau\} \cup \text{OP}(M) \).

Definition 5.6 (Synchronous GTS)
A Synchronous GTS that represents the synchronous composition of \( n \) STSs
is a tuple $\Sigma_S = (\Gamma^S, \Gamma_0^S, \sim^S)$, where $\Gamma^S$ is a set of configurations without queues $\gamma = (\bar{s}, \bar{g}, \bar{u})$, $\Gamma_0^S \subseteq \Gamma^S$ is a set of initial configurations with $\bar{u} = \bar{u}_0$, and $\sim^S \subseteq \Gamma^S \times \{A \cup \text{tick}\} \times \Gamma^S$ is a transition relation such that $(\langle \bar{s}, \bar{g}, \bar{u} \rangle, a, (\bar{s}', \bar{g}', \bar{u}')) \in \sim^S$, if:

- $a = \tau$ and for some $1 \leq i \leq n$, for some $(s_i, \phi, a, \Omega, Y, \vartheta, s'_i) \in R^i$
  
  $\bar{g}, \bar{u} \models \phi$, $\bar{s}' = \bar{s}[s'_i/s_i]$, $\bar{u}' = \bar{u}[Y \rightarrow 0]$, $\bar{g}' = \text{update}(\bar{g}, \Omega)$, $\bar{u}' \models L_1(\bar{s}')$;

- $a = \mu$, with $\mu \in \text{OP}(M)$, and for some $1 \leq i, j \leq n$, for some $(s_i, \phi_i, \overrightarrow{\mu}(\bar{x}_i), \Omega_i, Y_i, \vartheta_i, s'_i) \in R^i$, $(s_j, \phi_j, \overrightarrow{\mu}(\bar{x}_j), \Omega_j, Y_j, \vartheta_j, s'_j) \in R^j$
  
  $\bar{g}, \bar{u} \models \phi_i \land \phi_j$, $\bar{s}' = \bar{s}[s'_i/s_i, s'_j/s_j]$, $\bar{u}' = \bar{u}[Y_i \cup Y_j \rightarrow 0]$, $\bar{g}' = \text{update}(\bar{g}, \Omega_i \cup \Omega_j \cup \{\bar{x}_j := \bar{x}_i\})$, $\bar{u}' \models L_1(\bar{s}')$;

- $a = \text{tick}$, $\bar{s}' = \bar{s}$, $\bar{g}' = \bar{g}$, $\bar{u}' = \bar{u} + d$ for some $d \in T$, $\bar{u}' \models L_1(\bar{s})$, and
  
  - $\forall 1 \leq i \leq n$, $\forall (s_i, \phi, \tau, \Omega, Y, \vartheta, s'_i) \in R^i$, if $\bar{g}, \bar{u} \models \phi$, then $\vartheta = \text{false}$;
  
  - $\forall \mu \in \text{OP}(M)$, $\forall 1 \leq i, j \leq n$, $\forall (s_i, \phi_i, \overrightarrow{\mu}(\bar{x}_i), \Omega_i, Y_i, \vartheta_i, s'_i) \in R^i$, $\forall (s_j, \phi_j, \overrightarrow{\mu}(\bar{x}_j), \Omega_j, Y_j, \vartheta_j, s'_j) \in R^j$, if $\bar{g}, \bar{u} \models \phi_i \land \phi_j$ then $\vartheta_i \land \vartheta_j = \text{false}$.

That is, the internal action of some participant is performed, two partners perform an interaction action, or time passes if there are no other urgent actions enabled. A run of the synchronous composition is a sequence $\gamma_1, a_1, \gamma_2, a_2, \ldots$, such that $(\gamma_i, a_i, \gamma_{i+1}) \in \sim^S$.

Let us define a *conversation* of the composition as a sequence of communications performed during the execution [56]. That is, $\omega_c = \mu_1, \mu_2, \ldots$, with $\mu_i \in \text{OP}(M)$, is a conversation if there is a run $\omega = \sigma_1, a_1, \sigma_2, a_2, \ldots$, and for each $i > 0$ there exists $j \geq i$, s.t. $a_j = \overrightarrow{\mu_i}(\bar{x})$. We denote a conversation set of the composition as $\Xi_c$. A conversation of a synchronous composition is merely a sequence of message types occurred on the run.
When the verification properties are defined on the set of conversations, and the composition appears to be complete under $\Delta_1$ model, then one can use the synchronous product for the composition analysis.

**Theorem 5.2** Let $\Sigma_{\Delta_1}$ and $\Sigma_S$ be a complete composition of $n$ STSs under $\Delta_1$ model and their synchronous product respectively, with the corresponding conversation sets $\Xi_c(\Sigma_{\Delta_1})$ and $\Xi_c(\Sigma_S)$. Then $\Xi_c(\Sigma_{\Delta_1}) = \Xi_c(\Sigma_S)$.

**Proof.** First we prove that $\Xi_c(\Sigma_{\Delta_1}) \subseteq \Xi_c(\Sigma_S)$.

Consider a conversation $\omega_c = \mu_1, \mu_2, \ldots \in \Xi_c(\Sigma_{\Delta_1})$. By definition, there is a sequence $\gamma_1, a_1, \gamma_2, a_2, \ldots$, such that for each $j \geq 0$, $\gamma_j \xrightarrow{a_i} \gamma_{j+1}$, and for each $i \geq 0$ there exists $j \geq i$ such that $\overrightarrow{\mu_i} = a_j$. We will show that also in the synchronous composition $\Sigma_S$, $\gamma_j$ reaches $\gamma_{j+1}$ through $\overrightarrow{\mu}_i$ and a (possibly empty) sequence of internal transitions.

Consider some $k$ such that $\overrightarrow{\mu}_{i+1} = a_k$. Since the queue of $\Delta_1$ is bounded to 1, and the composition is complete, there should be some $j < l < k$ such that $\gamma_l \xrightarrow{\overrightarrow{\mu}_i} \gamma_{l+1}$. Transitions from $j + 1$ to $l$ (if any) are internal. If the reception of $\mu_i$ is not possible in a local state of $\gamma_j$, then there exist a subsequence of internal transitions in a sequence from $j + 1$ to $l$ that leads to a state where it is enabled: $\gamma'$. Since internal transitions of different STS are independent, $\gamma'$ is reachable from $\gamma_j$. In $\gamma'$ the actions $\overrightarrow{\mu}_i$ and $\overrightarrow{\mu}_i$ may be fired subsequently leading to a state $\gamma''$. Analogously, the remaining transitions in a sequence from $j$ to $k$ are internal transitions and may be fired in $\gamma''$. Obviously, a sequence $\gamma_j, \ldots, \gamma', \mu_i, \gamma'', \ldots, \gamma_k$ is fireable also in $\Sigma_S$, and produces the same conversation ($\mu_i$). Therefore, any conversation of $\Sigma_{\Delta_1}$ is also a conversation of $\Sigma_S$.

In order to see that $\Xi_c(\Sigma_S) \subseteq \Xi_c(\Sigma_{\Delta_1})$, consider a conversation $\omega_c$ and a corresponding sequence $\gamma_1^S, a_1, \gamma_2^S, a_2, \ldots$ fireable in $\Sigma_S$. Every transition $\gamma_j^S \xrightarrow{a_j} \gamma_{j+1}^S$ may be represented by a corresponding transition of $\Sigma_{\Delta_1}$, if $a_j = \tau$, or by a sequence of send and receive transitions of $\Sigma_{\Delta_1}$, if $a_j = \mu$. Obviously, such a sequence produces the same conversation $\omega_c$. 
As a result, $\Xi_c(\Sigma_S) = \Xi_c(\Sigma_1)$. □

**Implementation requirements.** Due to the strong hypotheses on the synchronizable communication model, the kinds of systems, for which the model is adequate, are also subject to restrictive hypotheses on the kinds of interactions that can occur. Consequently, the compositions, for which this model is proved to be adequate, are very robust and exhibit the same behavior on any BPEL engine implementation. For this reason, the synchronizable model is the less demanding on the underlying middleware among the ones studied here.

**Locally Ordered Asynchronous Communications.**

This model is used in some works for the representation of Web service compositions (see e.g. [56]). Each participant is equipped with separate queue storing messages from all the partners.

**Definition 5.7** A locally ordered asynchronous communication model for the composition of $n$ STSs is a model $\Delta_{lo} = \langle B, \mathcal{L}_M, \mathcal{L}_O \rangle$, with $n$ ordered queues, $b_i = \infty$, and $\forall \alpha$, s.t. $\mu_\alpha \in I^i$. $\mathcal{L}_M(\alpha) = q_i$.

This model is more general than the synchronizable model:

$$\Delta_1 \subseteq \Delta_{lo}.$$

Indeed, this model is required for describing the composition scenario of Example 2.

**Implementation requirements.** This communication model requires that messages are queued on a process-by-process way. This policy for managing queues is a reasonable and easy to implement, and it provides a good compromise between the complexity of the implementation and the class of examples it is able to cover.
5.3. HIERARCHY OF MODELS

Mutually Ordered Asynchronous Communications.

In this model, a pair of queues is defined for each pair of processes, where each queue represents one direction of interaction between these processes. This model, described in [19], provides a natural representation of communicating BPEL processes since each process explicitly distinguishes each of its partners. The main feature of this model is that each pair of communicating processes preserves the order of partners’ events. In other words, the order of receptions is equivalent for each pair of processes.

**Definition 5.8** A mutually ordered asynchronous communication model of \(n\) STSs is \(\Delta_{mo} = \langle B, L_M, L_O \rangle\), with \(n^2 - n\) ordered queues denoted as \(q_{i,j}\) \((i \neq j)\), s.t. \(b_{i,j} = \infty\), and \(\forall \alpha. \mu^i_\alpha \in I^i \land \mu^j_\alpha \in O^j\) iff \(L_M(\alpha) = q_{i,j}\).

This model is clearly more general than the synchronous model. It is also more general with respect to locally ordered asynchronous model. Indeed, the input messages of the particular process are stored potentially in several queues instead of only one. Therefore, the alphabets of these queues are smaller, and more actions are fireable (Theorem 5.1). This model is required for the composition scenario described in Example 3.

**Implementation requirements.** As in the case of previous model, the policy for managing queues dictated by the mutually ordered communication model is a reasonable and easy to implement. In particular, it allows for separation of messages from different partners, which reflects the vision adopted by BPEL.

5.3.4 Summary

The parametric communication model allows us to define a hierarchy of different communication patterns. While potentially infinite number of
different models may be defined, we consider only several of them that are widely used in practice. These models form the following hierarchy:

\[ \Delta_1 \subseteq \Delta_{lo} \subseteq \Delta_{no} \subseteq \Delta_{MG}. \]

The MG-model is always an upper bound of any hierarchy, and we use this fact in the adequacy analysis presented below.

## 5.4 Building an Adequate Model

We now present an approach for the analysis of compositions of Web services. In this approach, we incrementally pass through the models starting from the synchronizable until the least general adequate model is found for the given composition scenario. As we will see in the evaluation experiments, this allows not only to find a proper model of communication for the scenario but also to perform the analysis more efficiently. Indeed, if the model is shown to be adequate for the given composition, and the composition behaves correctly, then it will be correct also under more general models.

The number of models that we could consider in our approach is potentially infinite. Here, we assume to have fixed a finite set of models that we consider interesting for the analysis (this could be for instance the sequence of models we have introduced in the previous section). We assume moreover that the simulation relation defines a total order on these models, and that the MG-model belongs to the set (and is hence its upper bound). More precisely, the algorithm of the approach is as follows:

1. take a sequence of models \( \Delta_1, \ldots, \Delta_n = \Delta_{MG} \) such that \( \Delta_i \subseteq \Delta_{i+1} \);
2. analyze the models until the adequate one is found: \( \Sigma_{\Delta_i} \approx \Sigma_{\Delta_{MG}} \);
3. the composition is checked for completeness (i.e., the queues are empty in the terminal states of the composition) and bounded growth.
5.4. BUILDING AN ADEQUATE MODEL

When an adequate communication model is identified, and the composition is shown to have queues with a bounded growth, the obtained reachability graph may be used as a basis for further verification tasks.

5.4.1 Algorithm

The algorithm is used to give an answer for the following questions: (i) the model under consideration is adequate for the description of the given composition; (ii) the composition is complete and has queues with a bounded growth.

The algorithm is presented in Alg. 1. The outcome of the algorithm is the constructed reachability graph representing the composition, if the model is adequate, or the witness of the fact that the model is not adequate.

In order to terminate the search when the composition has unbounded growth, we use a special parametric relation $U$ that defines an ordering between states. We write $U(\gamma, \gamma')$ to denote that the $\gamma$ precedes $\gamma'$. We also write $U \neq (\gamma, \gamma')$ to denote the strict precedence, i.e., that $U(\gamma, \gamma')$ and not $U(\gamma', \gamma)$. We discuss the relation in details in the following section.

The behavior of the algorithm is the following.

- A set of containers to store the reachability graph structure and the “bad” states (i.e. unbounded or incomplete) is defined (lines 1-5).

- The recursive procedure $explore$ that implements a DFS-algorithm is defined (lines 7-24) and called for initial state (line 6).

- The current state is added to the stack and a set of fireable actions from this set is extracted (line 10). If this set is not equivalent to the set of actions fireable under MG-model ($fireable_{MG}$), then the algorithm terminates since the selected model is not adequate (11).
Algorithm 1 Composition adequacy check

1: States := nil; \{Stack of states\}
2: Visited := nil; \{Set of visited states\}
3: Transitions := nil; \{Set of transitions\}
4: IS := nil; \{Set of incomplete states\}
5: US := nil; \{Set of unbounded states\}
6: explore(\(\gamma_0\));

7: procedure explore(\(\gamma\))
8: push(\(\gamma\), States);
9: current := \(\gamma\);
10: Fireable := fireable(current); \{fireable actions\}
11: if Fireable \(\neq\) fireable\(_{MG}\)(current) then terminate;
12: Transitions := Transitions \(\cup\) Fireable;
13: if Fireable \(\neq\) \(\emptyset\) then
14: for all transition \(\in\) Fireable do
15: \(\gamma' :=\) transition.target;
16: if \(\gamma' \notin\) States \(\cup\) Visited then
17: if \(\exists\) \(\gamma'' \in\) States s.t. \(U(\gamma'', \gamma')\) then
18: \(US := US \cup \{current\}\);
19: else explore(\(\gamma'\));
20: else if \(\neg\) emptyChannels(current) then
21: \(IS := IS \cup \{current\}\);
22: Visited := Visited \(\cup\) \{current\};
23: pop(s, States);
24: end procedure

- If the set of actions is not empty, then we analyze the target state \(\gamma'\) of each action (14-19).

- If the state is fresh then we check the queue unboundedness by looking for a cycle that increments queue content (17-18). If there is no such a cycle, we call explore procedure recursively.

- If the set of fireable actions is empty then we reached the leaf of the search tree and just check that all the messages are consumed (20-21).
Whenever a leaf state with non-empty queue content is found, it is added to the special container *incomplete*.

- The current state is thus explored; we add it to the set of visited states, and remove from the stack.

### 5.4.2 Dealing with Data

The presence of data and time in the composition model allows for a potentially infinite state space. Moreover, infiniteness of data values in messages content leads to the infiniteness of the alphabet on which the queues are defined. This prevents the direct application of automated techniques for the analysis of an arbitrary composition. In particular, the adequacy analysis algorithm presented above is not guaranteed to terminate.

One of the ways to address the problem is as follows. First, the algorithm may be executed on a *skeleton* of the composition, i.e., the model of the composition, where the data and time components are omitted. The state space in this case is finite and the algorithm always terminates. Second, it is necessary to define certain criteria that would allow to extend the obtained results also on a full composition model. Third, the constructed reachability graph is populated with the data- and time-related part for the further analysis. We will discuss the skeleton-based adequacy analysis in the following.

**Skeleton-based Composition Analysis**

In the skeleton of the composition, the time and the data (i.e. clocks, functions, variables, message contents) are abstracted away: $\mathcal{X} = \emptyset$, $\mathcal{F} = \emptyset$, $\mathcal{V} = \emptyset$, $\forall \alpha \in M \ \mathrm{val}(\alpha) = \epsilon$. The set of messages of the skeleton is a set of service operations (types) defined in the original composition: $\{\text{op}(\alpha) \mid \alpha \in M\}$. 

100
A skeleton of the composition is defined as a composition of the skeletons of the corresponding STSs under the skeleton of the corresponding communication model $\Delta$.

**Definition 5.9 (Skeleton of the STS)**

A skeleton $[\Sigma]$ of the STS $\langle S, S_0, V, X, A, R, L_I \rangle$, is a tuple $\langle S, S_0, A, R' \rangle$, where $R' \subseteq S \times A \times S$ is a skeleton transition relation, s.t. $(s, a, s') \in R'$ if there exists a transition $(s, \phi, a, \Omega, Y, \vartheta, s') \in R$.

The skeleton of the communication model may be defined by replacing alphabets of messages with the corresponding message types. This, however, may lead to a situation, where there are two (or more) queues with intersecting alphabets, or it is not clear whether the resulting queue is ordered or not. Therefore, such a communication model should be discarded. In this case we say that the communication model is not *well-formed*. Let us denote the set of message types of a queue as $\text{OP}(q_i) = \{\text{OP}(\alpha) \mid \alpha \in M_i\}$.

**Definition 5.10 (Well-formed Communication Model)**

We say that the communication model $\Delta = \langle B, L_M, L_O \rangle$ is well-formed, if for each pair of queues $q_i$ and $q_j$ their sets of message types are disjoint:

$$\text{OP}(q_i) \cap \text{OP}(q_j) = \emptyset.$$ 

In the following we will consider only well-formed communication models.

**Definition 5.11 (Skeleton of the Communication Model)**

Skeleton of the communication model $\Delta$, is the communication model $[\Delta]$, such that there exists a queue $q' \in [\Delta]$, with alphabet $M'$, bound $b'$ and ordering $L'_O$, if there exists a queue $q_i \in \Delta$, with $\text{OP}(q_i) = M'$, $b_i = b'$, and $L_O(i) = L'_O$, such that $\text{OP}(q_i) \cap \text{OP}(q_j) = \emptyset$, for any $j \neq i$.

The approach based on the composition skeleton analysis consists of the following steps. First, the skeleton is being constructed using the above
algorithm. Second, if the system is shown to be bounded, the resulting reachability graph is enriched with data and time, and passed to the model checker for more refined analysis (e.g., completeness verification, properties verification, etc.).

This approach gives only partial result: indeed, the skeleton of the composition may be unbounded, while this is not a case for the original model. In this case more refined analysis should be applied. On the contrary, as we will see later, if the skeleton is bounded, the original model is also bounded.

Since the skeleton model does not contain the data and time, the control part of the composition is finite (provided that the sets of program locations of participating STSs are finite). The only possibility to have an infinite reachability graph is when there is an infinite sequence of nodes of reachability tree with an infinite number of different states (Koenig’s lemma). This, in turn, is possible if and only if the number of messages contained in channels can grow unboundedly.

**Proposition 5.3** The reachability graph of the skeleton of the composition $RG([\Sigma_{\Delta MG}])$ is finite if and only if the skeleton is bounded [112].

The termination of the algorithm relies on the relation $U$ between states defined as follows:

**Definition 5.12** For two nodes of the reachability tree labeled with configurations $\gamma$ and $\gamma'$

$$U(\gamma, \gamma') \iff \sigma(\gamma) = \sigma(\gamma') \land C(\gamma) \leq C(\gamma')$$

Two states are in the relation iff the observable state is the same and the queue content increased. The following theorem justifies the correctness and termination of the adequacy analysis algorithm for the skeleton of the composition:
Theorem 5.3 The skeleton of the composition under the MG-model is unbounded if and only if there exist two nodes of the reachability tree $\gamma$ and $\gamma'$, such that $\gamma \xrightarrow{\omega}^* \gamma'$, and $U \neq (\gamma, \gamma')$.

Proof. $\Rightarrow$ direction. Since the skeleton is unbounded, the reachability graph is infinite. This means that there exists an infinite sequence of nodes of the reachability tree with infinite number of different states. We will show that it contains two nodes $\gamma$ and $\gamma'$ such that $\gamma \xrightarrow{\omega}^* \gamma'$, and $U \neq (\gamma, \gamma')$.

Since the control part of the skeleton is finite, we can extract a finite number of subsequences with identical control parts. At least one subsequence is infinite, and we denote one of them as $\omega_\infty$. Fix some message type $\mu_1$. Since the content length is bounded from below, and $\omega_\infty$ is infinite, it contains an infinite subsequence, in which the number of messages of type $\mu_1$ is increasing monotonically: $\omega_\infty^1$. Starting from this subsequence, we repeat the construction for other message types. As a result, we obtain an infinite subsequence, where the numbers of messages of all message types grow infinitely: $\omega_\infty^n$. Therefore, there exist two nodes $\gamma$ and $\gamma'$ such that $C(\gamma) \leq C(\gamma')$. Since the observable state of all the nodes of $\omega_\infty$ are identical, $U(\gamma, \gamma')$. To see that $U \neq (\gamma, \gamma')$ is possible, note that the number of different states is infinite.

$\Leftarrow$ direction. There are nodes $\gamma$ and $\gamma'$, reachable from $\gamma$, such that $U \neq (\gamma, \gamma')$. Let $\omega = \gamma, a_1, \gamma_2, \ldots, a_n, \gamma'$ be a sequence leading from $\gamma$ to $\gamma'$. Note first, that the action $a_1$ is enabled also in $\gamma'$, leading to some $\gamma'_2$, such that $U \neq (\gamma_2, \gamma'_2)$. Indeed, the transition is defined independently from the queue structure, leading to a configuration with the same observable state. If it is an internal transition, it is enabled independently of the queue content. Since the queue is unbounded, any output transition is enabled too, incrementing the length of the queue. Since $C(\gamma) \leq C(\gamma')$, any input transition is also enabled. Therefore, $\gamma' \xrightarrow{a_1} \gamma'_2$, and $U \neq (\gamma_2, \gamma'_2)$.
5.4. BUILDING AN ADEQUATE MODEL

This construction chain may be infinitely prolonged, leading to a sequence with infinitely growing queue contents, and therefore infinite number of different nodes. As a result, the skeleton has unbounded growth. □

Note, however, that this result does not hold for compositions under an arbitrary communication model. In particular, in [19] it was shown, that the reachability problem in the composition of communicating finite state machines (CFSMs) is undecidable. The formalisms of CFSM composition is a subclass of GTS model presented here, with the communication model similar to mutually ordered asynchronous composition $[\Delta_{mo}]$.

Unboundedness of the skeleton does not necessarily mean unboundedness of the full composition. To see this, consider an example presented in Fig. 5.4. While in the composition only five messages may be sent, the skeleton is unbounded. On the contrary, boundedness and adequacy of some model of the skeleton, implies also boundedness and adequacy in the full composition, thus justifying the skeleton-based approach.

**Theorem 5.4** If the skeleton of the composition under the MG-model is bounded, then the composition under the MG-model is also bounded.

**Proof.** Assume that the skeleton of the GTS is bounded, while the GTS is not. The unboundedness of the composition implies that the content of the queue grows infinitely. Consider an arbitrary configuration of GTS
CHAPTER 5. ANALYSIS OF COMMUNICATION MODELS

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure5_5.png}
\caption{Inadequate model is adequate for the skeleton}
\end{figure}

\[ \langle \bar{s}, \bar{g}, C, \bar{u} \rangle \text{ and an analogous state of the skeleton } \langle \bar{s}, C' \rangle. \]

From the definition of the skeleton of the MG-model follows that \(|w'| = |w|\). Since the skeleton is bounded, then for each reachable state there exists some bound \(K \geq 0\), such that \(|w'| \leq K\). By the unboundedness assumption, for any \(K > 0\) there exists some state \(\langle \bar{s}, \bar{g}, C, \bar{u} \rangle\), in which \(|w| > K\). The state is reachable in GTS. In order to see that the corresponding state \(\langle \bar{s}, C' \rangle\) is reachable in the skeleton, note that each action fireable in a state of the \(\Sigma_{\Delta_MG}\) is also fireable in the corresponding state of the skeleton. Therefore, \(|w'| > K\), which contradicts with the above assumption. \(\Box\)

The adequacy of the communication model skeleton for the composition skeleton does not directly implies the adequacy of the model in the composition itself. The reason is that different messages, while indistinguishable in the skeleton, not necessarily satisfy the same conditions or lead to the same assignments. Consider an example of the composition represented in Fig. 5.5. If the model is defined in such a way that the messages of type \(b\) belong to the same ordered queue, then in the composition a state labelled as \(S2\) is not reachable. It is easy to see that this state is reachable in the composition under MG-model as well as in the skeleton.

In order to define a condition that would ensure analogous result, an
addition check is needed. In particular, it is needed to discard models, in which a configuration containing two different messages of the same type in ordered queue is reachable. More precisely, the following theorem defines the required analysis extension. We denote the number of messages of type \( \mu \) stored in a queue \( q \) as \( |q|_\mu \).

**Theorem 5.5** If \([\Delta]\) is adequate for \([\Sigma_\Delta]\), and each reachable state of \([\Sigma_\Delta]\) does not contain more than one message of the same type in the same ordered queue, then \( \Delta \) is also adequate for \( \Sigma_\Delta \).

**Proof.** Assume that the model \( \Delta \) is not adequate for the composition. That is, \( \Sigma_\Delta \not\approx \Sigma_{\Delta_MG} \). However, due to the fact that \( \Delta \subseteq \Delta_MG \), we have that \( \Sigma_\Delta < \Sigma_{\Delta_MG} \). This implies, in turn, that some initial configuration \( \gamma_{0MG} \) of \( \Sigma_{\Delta_MG} \) is not simulated by any initial configuration of \( \Sigma_\Delta \). Consider an initial configuration \( \gamma_0 \) of \( \Sigma_\Delta \) with the same observable part as in \( \gamma_{0MG} \). By the assumption, there is a sequence \( \omega \) of (zero or more) transitions, configurations \( \gamma \) and \( \gamma_{MG} \), such that \( \gamma_{0MG} \xrightarrow{\omega} \gamma_{MG} \), \( \gamma_0 \xrightarrow{\omega} \gamma \), \( \sigma(\gamma) = \sigma(\gamma_{MG}) \), and \( \gamma_{MG} \not\leq \gamma \). This, in turn, implies that there exist \( a \in A \) and \( \gamma_{1MG} \) of \( \Sigma_{\Delta_MG} \), such that \( \gamma_{MG} \xrightarrow{a} \gamma_{1MG} \), and for any \( \gamma_1 \) of \( \Sigma_\Delta \), either \( \gamma \not\rightarrow a \gamma_1 \), or \( \sigma(\gamma_1) \neq \sigma(\gamma_{1MG}) \).

Consider a state \( \gamma'_{MG} = \langle \bar{s}, C'_{MG} \rangle \) of \([\Sigma_{\Delta_MG}]\) corresponding to \( \gamma_{MG} \). Obviously, this state is reachable from some initial state \( \gamma'_0_{MG} \) corresponding to \( \gamma_{0MG} \) through the same sequence of actions. Since \([\Sigma_\Delta] \approx [\Sigma_{\Delta_MG}]\), there exists a reachable state \( \gamma' = \langle \bar{s}, C' \rangle \) of \([\Sigma_\Delta]\), such that \( \gamma' \approx \gamma'_{MG} \). Moreover, the state \( \gamma' \) is unique for the given sequence \( \omega \). We have to consider the following situations.

\begin{itemize}
    \item \( a = \tau \). Obviously, the action is fireable in \( \gamma \) iff it is fireable in \( \gamma_{MG} \), and the resulting configuration has the same observable part: \( \gamma \xrightarrow{a} \gamma_1 \) and \( \sigma(\gamma_1) = \sigma(\gamma_{1MG}) \).
\end{itemize}
• \(a = \overrightarrow{\mu}\). That is, there exists \(\alpha\), with \(\text{op}(\alpha) = \mu\), such that \(C_{1MG} = C_{MG}.\alpha\). Since \(\sigma(\gamma) = \sigma(\gamma_{MG})\), then \(a\) is not fireable only if for some queue \(q_i\), \(\alpha \in M_i\), \(|w_i| = b_i\). Consider a state \(\gamma'\) of \([\Sigma_{\Delta}]\). \(a\) is fireable in \(\gamma'\), therefore there exists a queue \(q'_j\) of \([\Sigma_{\Delta}]\), \(\mu \in M'_j\), \(|w'_j| < b_j\). The amount of messages of the same type in the queues of \(\gamma\) and \(\gamma'\) is the same: \(|w'_j| = \sum_{\mu \in M_i} |w_i|\). Thus, \(|w'_j| \geq |w_i|\). By the definition of the skeleton of the composition model, \(b'_i = b_i\). Consequently, \(|w_i| \leq |w'_j| < b'_i = b_i\), which contradicts with the above assumption.

• \(a = \overleftarrow{\mu}\). That is there exists \(\alpha\), with \(\text{op}(\alpha) = \mu\), such that \(C_{MG} = \alpha.C_{1MG}\). Consider a state \(\gamma'\) of \([\Sigma_{\Delta}]\) reachable from \(\gamma'_0\) through the same sequence \(\omega\). \(a\) is fireable in \(\gamma'\), thus \(C' = \mu.C'_{1}\). If the corresponding queue is unordered, and the action is fireable in \(\gamma_{MG}\), then obviously \(a\) may be fired in \(\gamma\) leading to the state with the same control part. Therefore, we have to consider only the case of ordered queue. By the above assumptions we have that

- some message \(\alpha_1\) of type \(\mu\) is in the head of some (ordered) queue, but the transition is not fireable or does not lead to the state with observable part \(\sigma(\gamma_{1MG})\). In the former case the guard \(\phi\) does not evaluate to true, in the latter case some variables are assigned to values different from those in \(\sigma(\gamma_{1MG})\). This is only possible if \(\alpha_1 \neq \alpha\). Therefore, there exist more than one message of type \(\mu\) in the queue contents, which contradicts with the assumption.

- there is no queue with the message of type \(\mu\) in the head. In the state of the skeleton, however, there is a message of this type in the head of the corresponding queue. Therefore, there exists some queue \(q_i\) of \(\Sigma_{\Delta}\), and some \(\alpha_1\), with \(\text{op}(\alpha_1) = \mu\), such that \(q_i = w_{i1}.\alpha_1 \ldots\), where prefix \(w_{i1}\) is not empty, and all the messages in \(w_{i1}\) were consumed in skeleton when executing \(\omega\). Since the same
sequence of actions is executed in $[\Sigma_\Delta]$ and $\Sigma_\Delta$, then the messages in the prefix were consumed instead of some other messages $w_{i2}$ of the corresponding types. Such a situation is possible only if there is another queue, which accepts the messages of the same types. This, however, is possible only if the model is not well-formed. □

We remark, that this check can be easily introduced in the adequacy analysis algorithm. In every visited state it is necessary to check that there are no two messages of the same type stored in some queue (of the skeleton). If this happens, it is needed to check whether the corresponding queue (or queues) of the model $\Delta$ is ordered or not, which can be done statically.

5.5 Discussion

We presented an approach to the formal analysis of communication models for the verification of Web service compositions. The approach is based on a parametric model of the composition described in Chapter 3. This approach allows for defining a rich hierarchy of various communication patterns that appear in practical settings. Using this hierarchy, we introduce an algorithm that is able to identify the simplest adequate communication model, i.e., the model that fully describes the behavior of the given composition and puts less requirements on the implementation. Moreover, as we will see in Section 8.2.1, finding the right model can lead to a substantial improvement of the analysis performance.

The presented approach follows in a certain sense the line suggested in [56]: given a set of processes, determine whether the synchronous model of communications may be applied for the analysis without loosing behaviors. For this purpose the authors present a set of conditions that enable to deduce this property. Our approach has a broader sense: it can check
the applicability of a wide range of models, and control queue boundedness. Another limitation of [56] with these regards is the usage of a fixed communication model, where one FIFO queue is associated to a service. This model is, however, not realistic to a wide range of protocols. In particular, the queue model used by several BPEL engines is different, their implementation is more complex. As a result, the class of compositions that we are able to model and analyze is substantially larger than those covered by other verification approaches.

The analysis of distributed systems under various models of message exchange (e.g., perfect/lossy, bounded/unbounded, ordered/unordered) has been widely studied in the literature (see [2] for a survey of the related works). Depending on the specific mechanism used, the complexity of the analysis tasks (reachability, boundedness, model checking) varies up to undecidability, and hence specific techniques are required. Our analysis algorithm is inspired by the unboundedness test proposed in [75]. In that work the authors propose a generic but not necessarily complete algorithm for checking queue unboundedness in a given system. While the question there is whether a given system property is affected by a given communication medium, in our analysis we are looking for a set of properties that are affected by some communication medium, such as possible unboundedness, sensitivity to reordering of messages, etc. Even if sometimes only partial answers are provided (the boundedness problem is undecidable in general), the analysis allows to obtain a certain characterization of an application related to the asynchronism of interactions.

This sensitivity of the composition behavior to the usage of communication models with different characteristics was used in our work presented in [82]. In that work we analyzed the problem of the choreography realizability, initially described in [55]. The problem is formulated as follows: given a global model (choreography), it is possible to extract local ones
(service specifications) such that their composite behavior corresponds to the global specification. The difference appear due to a possible concurrency of requests, while in the global model the interactions are atomic. The notions of “realizability” are given hierarchically, and are aligned with communication models that express the corresponding requirements. The requirements may express complete correspondence, correspondence between emission and reception sequences, correspondence between mutual partner protocols, etc. By checking adequacy of such models, we can identify particular realizability requirements the composition is sensitive to.

In the current version of the approach we assume that the hierarchy of the communication models is defined before the analysis. However, it is possible to avoid the usage of an explicit hierarchy, and construct a minimal adequate model on-the-fly. This would allow to improve the analysis performance, to obtain more precise answer, and to determine particular reorderings that are critical for the differences in behavior. We leave this issue as a future work.
Chapter 6

Data Flow Analysis

The exchange and management of business data in service compositions using XML-based standards is one of the most important capabilities of the Web service technology. In data-intensive applications the data flow is as critical for the behavioral analysis as the control flow, since the application execution is driven by the manipulated information. In most of the existing approaches to the analysis of service compositions the data-related issues are abstracted away, hence potentially invalidating the analysis results.

Analysis of data in Web service compositions has to deal with several problems that restrict its applicability both for what concerns the efficiency and the quality of the results. First, the data domains operated by the Web service specifications are often infinite (e.g., user-defined type in XML documents), and the semantics of data operations is complex (e.g., the XPath expressions used in BPEL specifications). This invalidates the usage of traditional formal analysis techniques relying on simple and finite system representations. Second, the internal details on service implementations and data management usually are not revealed to the service users, the service descriptions contain only partial information with these regards (e.g., interfaces, preconditions and effects, conversation specifications, policies). This makes the non-determinism and incompleteness of the data flow
information an inherent property of the composition models.

A variety of techniques, and in particular abstraction refinement, were proposed to address the first problem, i.e., the infiniteness and complexity of the data flow constructs. These techniques allow for considering in the analysis only relevant data-related properties. The partial knowledge and non-determinism on the data flow in the Web service domain, however, make these techniques insufficient, and require novel approaches to address a wider problem. Instead of providing a yes/no answer for the property verification, it is crucial to determine under which assumptions about unknown service functionality the property may be satisfied/violated. As yet, this problem is not addressed by the existing approaches and is a challenging research line.

In this chapter we discuss a novel abstraction-based approach to the behavioral analysis of Web service compositions in presence of data. We foresee such an analysis as an iterative process, where the composition model is incrementally refined. The refinement is achieved by elicitation of the assumptions about unknown data transformations and functions that ensure the correctness/violation of the analyzed property. As a result, we obtain a model where the initial composition specification is enriched with a set of requirements that are crucial for the system correctness, and where all the expected properties have been formally verified. While these requirements are discovered and collected during the static, design-time verification of the compositions, they may then be further exploited for the dynamic, run-time analyses that rely on monitoring techniques.

The presented analysis relies on data abstraction techniques, and a formal model for the data flow constraints specification. The two presented models of abstraction allow to define satisfiability bounds for a verified property, and the data flow constraints are needed to reduce the uncertainty, and hence to refine the analyzed system.
6.1 Case Study: Extended Loan Approval

In order to illustrate the data-related problems of composition analysis we consider a (modified) Loan Approval case study. The nominal scenario may be summarized as follows. The user requests a loan approval specifying a desired amount and the Loan Approval (LA) process is aimed to iteratively look for an amount that may be safely approved for the user. Each iteration is performed as follows. The current amount is being checked. If the amount is not large, the Assessor service is called, otherwise the Approver service is used. If the Assessor considers the amount to be risky, then it is also passed to the Approver, otherwise the amount is approved and the loop terminates. If the Approver provides a negative answer, the amount is reduced and the iteration continues. We model the composition as a set of BPEL specifications that represent the behaviors of every participant. A conceptual model of the composition scenario is represented in Fig. 6.1.
6.1.1 Ignoring Data Flow

Ignoring the data even in this example leads to the following problems in the analysis.

First, scenarios are possible that never happen in a real system. Indeed, the body of the loop is implemented as a flow construct with synchronization links between activities. As soon as the link condition evaluates to true, the target activity is ready to start. Since the link conditions leaving the same activity are mutually exclusive, both target activities can not be fired and the behavior is correct. However, if the data is ignored, both Assessor and Approver services may be invoked and the approval fails. Moreover, if we verify the property that “it is possible for the customer to make a request that LA accepts without involving Approver” (which is expected to be possible for small amounts), the data-less analysis would say that the property does not hold. Another property that is violated when the data is ignored is the following: “for any request the LA service should provide a result”.

Second, the data-less analysis may hide potential problems that can be revealed only in the presence of data. In our example, if the answer of the Approver service is positive, the approved amount should be returned to Customer. In the specification, however, the value of the result variable is returned instead. It is easy to see that a wrong value may be returned to Customer, since this variable is assigned only initially, and is not changed during the execution of the loop. In order to fix this problem an operation that assigns the value of the amount variable to the value of the result variable has to be performed upon receiving the answer from the Approver service as it is done if the loan is approved without intervention of the Approver in case of low risk. Clearly, this problem may not be revealed if the data is ignored.
6.1.2 Abstractions-based Representation

The presented case study may not be directly verified using traditional formal techniques. Indeed, the data variables, being manipulated in this scenario have infinite domains, such as strings, integers, structures, and aggregations of them. Therefore a finite representation of the system is not possible. A predicate abstraction approach [63] allows to avoid this problem by considering only relevant data flow properties and ignoring the others. For instance, in the above case study it is not needed to enumerate all the possible amounts that the user may request. Instead, it might be sufficient to evaluate the fact that the amount is big, which can be represented with one proposition.

The abstraction, however, is not always able to faithfully describe the real system. Indeed, the abstract model may over-approximate the real system, that is may contain some unrealizable behaviors. If one is interested to check that every behavior satisfies a property (universal property), then the violation of the property in an over-approximated abstraction does not imply the violation in a real system. Analogously, if the verification of the existential property (i.e. some behavior satisfies the property) is violated in under-approximation (i.e. the abstract system that contains less behaviors then the real one), it is not necessarily violated in the real system. This will be taken into account in the analysis framework through an interplay of over- and under-approximations during the verification process.

6.1.3 Incompleteness of Composition Specifications

The specification of Web service composition often hides certain detail on the internal service implementations, complicating the analysis of the specification. Indeed, if the internal algorithms implemented by the Approver and Assessor services are not known, then the abstract models of
the composition do not contain enough knowledge to infer validity of many properties. In particular, it would be impossible to check the possibility to eventually provide an answer, or that any scenario that starts with a sufficiently small amount leads to an approval.

In order to eliminate this lack of knowledge, the model should be enriched with assumptions on the internal implementation that should imply the correctness of the behavior. In other words, if we consider these internal operations as uninterpreted functions then the assumptions may be thought of as the preconditions/effects rules on the function values. For example, if Assessor service is expected to return low risk whenever the amount is small (e.g. less than 100 euro), the LA process is guaranteed to terminate correctly for such small requests.

In the following sections we present a formal approach that (i) allows for the definition of abstract composition model, (ii) enables the specification and verification of both universal and existential properties, and (iii) supports the definition and analysis of data flow properties reflecting assumptions on service internals.

### 6.2 Data Abstraction Model

The verification of an STS composition is not doable in general due to the fact that the data types are infinite and the functions are too complex to reason on. In order to be able to perform a finite state verification, we have to provide an abstraction of the underlying composition model that is finitely representable and allows to obtain certain answers for the verification queries. We now define two models for abstraction, namely a knowledge-based model (K-model) and a branching-based model (B-model).
6.2.1 Abstract Propositions

Both models we introduce in this work are *parametric* with respect to the set of propositions being considered in the analysis. These propositions may express facts about the values of the composition variables, relations between them, function values, etc. The propositions have the form of data expressions according to the definition presented in Section 3.2.1.

In the abstract model of the composition only a subset of all the possible propositions is considered. The selection of this subset defines the parametrization of the abstraction. When the set of considered propositions increases, the abstraction gets closer to the real system. This, however, increases also the computational cost of the verification. Therefore, the parameterization allows one to drive the verification process, starting from a small set and adding new propositions only when a refinement is needed. In the following we will denote the subset of the propositions considered in abstraction $A$ as $\mathcal{P}^A$.

The set of propositions used in some formula may contain elements not in $\mathcal{P}^A$. In order to define the satisfiability of the formula with respect to the abstraction, we have to give the formula a different interpretation. Intuitively, this interpretation requires that the formula is represented in Negative Normal Form (i.e., the negation is applied only to atomic propositions), and if a proposition $p$ appears positively (respectively, negatively) in the formula, then $p \in \mathcal{P}^A$ (resp., $(\neg p) \in \mathcal{P}^A$). In our prototype tool this interpretation is constructed automatically, by pushing negations inside the formula, and potentially extending the set of abstract propositions. An abstract interpretation of the formula $\phi$ is denoted as $[\phi]$.

**Proposition 6.1** $\Sigma \models [\phi]$ if and only if $\Sigma \models \phi$.

Given a set of propositions $\mathcal{P}^A$, a *valuation* is simply a mapping from $\mathcal{P}^A$ to $\{true, false\}$. We denote the valuation as $V$. We call the set of
global ground states compatible with the valuation the interpretation of the valuation, denoted as $\mathcal{I}(V)$. We say that the valuation is consistent, written as $\text{consistent}(V)$, if and only if $\mathcal{I}(V) \neq \emptyset$. We define the valuation of a propositional formula as follows.

**Definition 6.1 (Valuation of the propositional formula)**

*Given a propositional formula $\phi$, its valuation $V(\phi)$ is defined as follows:*

- if $[\phi] = p$, for some $p \in \mathcal{P}^A$, then $V(\phi) = V(p)$;
- if $[\phi] = \phi' \land \phi''$ then $V(\phi) = V(\phi') \land V(\phi'')$;
- if $[\phi] = \phi' \lor \phi''$ then $V(\phi) = V(\phi') \lor V(\phi'')$.

We write $V \models \phi$ when $V(\phi) = \text{true}$.

### 6.2.2 Abstract Composition

Analogously to the concrete composition model, the abstract composition model defines the evolutions of the system. However, since the variables, message contents, and functions are abstracted away, the communication model should be modified accordingly. That is, in the abstract communication model queue alphabets are finite, and the queues are bounded. Without loss of generality, we can allow the abstract propositions to be defined over queue content, and assume that the transitions are interpreted accordingly.

Abstract configuration in this model is a tuple $\gamma = \langle \bar{s}, V, C, \bar{u} \rangle$, where $V$ is the valuation of propositions. Since the variables, queue contents, and data domains are abstracted by the atomic propositions, the semantics of the composition model should be correspondingly defined. Similar to the definitions in Section 3.3.3, we say that a transition $t$ may be performed in an abstract configuration $\gamma$, if it is applicable, written as $\text{applicable}[\gamma](t)$.
The effect of the transition defines the modification of the abstract configuration according to the transition definition, denoted as \( \text{exec}[\gamma](t) \). We denote the applicability of the data condition \( \phi_D \) under the valuation \( V \) as \( \text{applicable}[V](\phi_D) \), and the effect of the assignment \( \Omega \) on the valuation \( V \) as \( \text{exec}[V](\Omega) \).

**Definition 6.2 (Abstract Applicability and Effect)**

A local transition \( t = (s_i, \phi, a, \Omega, Y, \vartheta, s'_i) \in \mathcal{R}^i \) is applicable in an abstract configuration \( \gamma = \langle \bar{s}, V, C, \bar{u} \rangle \), written as \( \text{applicable}[\gamma](t) \) if and only if:

- \( \bar{u} \models \phi_T \), \( \text{applicable}[V](\phi_D) \), where \( \phi_T \in \Phi_T \), \( \phi_D \in \Phi_D \), \( \phi = \phi_T \land \phi_D \);
- if \( a = \overrightarrow{\mu} \) then \( \forall \alpha \in M \) if \( \text{OP}(\alpha) = \mu \) then \( |C.\alpha| \leq B \);
- if \( a = \overleftarrow{\mu} \) then \( \exists C', \exists \alpha \in M \) such that \( C = \alpha.C' \) and \( \text{OP}(\alpha) = \mu \);

The effect of performing the transition \( t \) in the configuration \( \gamma \) is a configuration \( \gamma' = \langle \bar{s}', V', C', \bar{u}' \rangle \), written as \( \gamma' = \text{exec}[\gamma](t) \), such that:

- \( \bar{u}' = \bar{u}[Y \mapsto 0] \), \( V' = \text{exec}[V](\Omega) \), and \( \bar{s}' = \bar{s}[s'_i/s_i] \);
- if \( a = \overrightarrow{\mu} \) then \( \exists \alpha \in M \) s.t. \( \text{OP}(\alpha) = \mu \), \( C' = C.\alpha \);
- if \( a = \overleftarrow{\mu} \) then \( \exists \alpha \in M \) s.t. \( \text{OP}(\alpha) = \mu \), \( C = \alpha.C' \);
- if \( a = \tau \) then \( C' = C \).

We now give a definition of the semantics of an abstract STS composition corresponding to a concrete one. This definition is parametric with respect to the interpretation of the valuation of the propositions, and to the way the applicability and the effect of the transition are defined, which differ for the two abstraction models.

**Definition 6.3 (Abstract Global Transition System)**

An abstract global transition system \( \Sigma^A = (\Gamma^A, \Gamma^0_A, \rightsquigarrow^A) \), where \( \Gamma^A \) is a...
set of abstract configurations, $\Gamma^A_0 \subseteq \Gamma^A$ is a set of initial configurations with $\bar{u} = \bar{u}_0$, and $\rightsquigarrow^A \subseteq \Gamma^A \times \{A \cup \text{TICK}\} \times \Gamma^A$ is a transition relation such that:

- $\langle s, V, C, \bar{u} \rangle \xrightarrow{\text{TICK}} \langle s, V, C, \bar{u} + d \rangle$, if
  - $\forall 1 \leq i \leq n, \forall t \in \mathcal{R}^i$, if $\text{applicable}[\langle s, V, C, \bar{u} \rangle](t)$ then $\vartheta = \text{false}$;
  - $(\bar{u} + d) \models \mathcal{L}_I(s)$;

- $\langle s, V, C, \bar{u} \rangle \xrightarrow{a} \langle s', V', C', \bar{u}' \rangle$, if $\exists 1 \leq i \leq n, \exists t \in \mathcal{R}^i$ such that
  - $\text{applicable}[\langle s, V, C, \bar{u} \rangle](t)$;
  - $\langle s', V', C', \bar{u}' \rangle = \text{exec}[\langle s, V, C, \bar{u} \rangle](t)$;
  - $\bar{u}' \models \mathcal{L}_I(s')$.

As in case of the concrete model, the semantics of the abstract composition allows for two kind of transitions: timed transition, where only time passes, and instant transition, where an action is fired and a location is changed. A behavior of the abstract composition $\Sigma^A_\Delta$ is defined in the usual way (see Def. 3.9).

### 6.2.3 K-Model of Abstraction

In our framework we exploit this model to show that in any ground model corresponding to the abstract one there exists a run satisfying a certain property, i.e. to verify the existential properties. The model presented below defines the most “pessimistic” assumption on the composition.

In the K-model an abstract state of the composition is represented at the “knowledge level”. In other terms, a certain fact about the abstract variable values may be known to be true, or unknown. If the valuation of a certain fact is $\text{true}$, then it is it is also true in all the ground state represented by the valuation. Thus, the fact is “known” to be true. On
the contrary, if the valuation is false, the fact is “unknown” to be true, and may be true or false in some ground state.

**Definition 6.4 (K-Interpretation)** Given a proposition valuation $V$, its K-interpretation, denoted as $I^K(V)$, is the set of global ground states $\bar{g} \in I^K(V)$ s.t. $\forall p \in \mathcal{P}^A$, if $V(p) = true$ then $\Gamma_{\bar{g}}(p) = true$.

The following results immediately hold for any consistent valuation under the K-model.

**Proposition 6.2** For any $V$, if consistent($V$) then for any $p, \neg p \in \mathcal{P}^A$, if $V(p) = true$ then $V(\neg p) = false$.

**Proposition 6.3** For any propositional formula $\phi$, for any $V$, if $V \models \phi$ then $\forall \bar{g} \in I(V)$, $\Gamma_{\bar{g}}(\phi) = true$.

The K-model defines a “pessimistic” view of the abstraction by implementing the applicability and the execution of the transition relation as follows. We say, that the transition is applicable in the abstract state, if it is applicable in every corresponding ground state. Analogously, in the K-model the effect of the transition execution is the most conservative with respect to the set of facts that can be deduced.

**Definition 6.5 (Applicability and execution in K-Model)**

A data condition $\phi_D$ is applicable in $V$, written as applicable[$V$]($\phi_D$)$^K$, iff $\forall \bar{g} \in I(V)$, $\Gamma_{\bar{g}}(\phi_D) = true$.

The execution of a transition $t$ on $V$, denoted as exec[$V$]($\Omega$)$^K$, is a valuation $V'$ s.t. $V'(p) = true$ iff $\forall \bar{g} \in I^K(V)$, $\Gamma_{update(\bar{g},\Omega)}(p) = true$.

For the abstraction under K-model we also assume that the set of initial states is a singleton with $V(p) = false$, for each $p \in \mathcal{P}^A$ (no facts are known a priori).
6.2. DATA ABSTRACTION MODEL

Under the applicability condition defined as above, a situation is possible where the execution of the abstract model reaches a location with branching, and no transition is allowed since neither the condition nor its negation is “known” to be true. Such an execution is to be discarded in the verification since the concrete system can not terminate in such a location. Consider, for instance, the Loan Approval example. Since there is no information on the implementation of the Assessor service, the result of the invocation of assess function is undefined, and the valuation of the proposition risk = low (as well as its negation) is “unknown”. The process execution is blocked in this location and has to be discarded.

Let us define a verification problem of K-model of STS composition. We say that the K-model of STS composition satisfies the possibility $\phi$ if and only if there is a run of the model that satisfies the abstract interpretation $[\phi]$ of the formula. Analogously, we say that this model satisfies the assertion $\phi$ if there is no run that satisfies $[\neg \phi]$.

**Definition 6.6 (Satisfiability in K-Model)**

The possibility $\phi$ is satisfied in K-model, written as $\Sigma^K \models_P \phi$, iff there exists a run $\omega^K$ of $\Sigma^K$, s.t. $\omega^K \models [\phi]$.

The assertion $\phi$ is satisfied in K-model, written as $\Sigma^K \models_A \phi$, iff for each $\omega^K$ of $\Sigma^K$, $\omega^K \not\models [\neg \phi]$.

The following results immediately follow from the definition of the K-model. For the lack of space, we omit the formal proofs in this work.

**Theorem 6.1**

Given an assertion $\phi$, if $\Sigma^K \not\models_A \phi$, then $\Sigma \not\models_A \phi$.

Given a possibility $\phi$, if $\Sigma^K \models_P \phi$, then $\Sigma \models_P \phi$.

**Proof.** Assertion. By definition, $\Sigma^K \not\models_A \phi$ implies that for some $\omega^K$ of $\Sigma^K$, $\omega^K \not\models [\neg \phi]$. From the definition of the interpretation, applicability,
and execution of the K-model follows that there exists a run \( \omega \), such that the same actions are fired, and for each intermediate observable state \( \sigma_i = (\bar{s}, \bar{g}, \bar{u}), \) \( i \geq 0 \), corresponding to an abstract observable state \( \sigma^K_i = (\bar{s}, V, \bar{u}) \) of \( \omega^K, \bar{g} \in I(V) \). From Proposition 6.3 follows that if \( V(\phi) = true \), then \( \bar{g} \models \phi \). To see that the implication propagates also for an arbitrary LTL formula, notice that the formula is in the negative normal form. By structural induction, obtain \( \omega^K \models [\phi] \Rightarrow \omega \models \phi \). As a result, there exists a run \( \omega \) in a concrete system, such that \( \omega \models \neg \phi \), or \( \omega \not\models \phi \), that is, the assertion is violated in a concrete model.

**Possibility.** May be proved using similar arguments. □

### 6.2.4 B-Model of Abstraction

The B-model is aimed to express an “optimistic” view of the executions that a system can perform. Whenever a valuation of some fact can not be determined, both true and false values are assumed. In other words, the abstract state is split in this case in two sets, with the true value in one of them and false in another. Therefore, if the valuation of the proposition is false, the proposition is also false in all the ground states of \( I(V) \).

**Definition 6.7 (B-Interpretation)** Given a proposition valuation \( V \), its B-interpretation, denoted as \( I^B(V) \), is a set of global ground states \( \bar{g} \) such that for each \( p \in \mathcal{P}^A \), \( V(p) = \Gamma_{\bar{g}}(p) \).

The following results immediately hold for any consistent valuation under the B-model.

**Proposition 6.4** For any \( V \), if consistent(\( V \)) then for any \( p, \neg p \in \mathcal{P}^A \), \( V(p) = true \) iff \( V(\neg p) = false \).

**Proposition 6.5** For any propositional formula \( \phi \), for any \( V \), \( V \models \phi \) iff \( \forall \bar{g} \in I(V), \Gamma_{\bar{g}}(\phi) = true \).
The “optimistic” approach requires that a transition is applicable in the valuation $V$, if $V$ is not in conflict with the transition condition. The effect of the transition in the B-model is defined by the set of the valuations that are compatible with the effects of the transition.

**Definition 6.8 (Applicability and execution in B-Model)**

A data condition $\phi_D$ is applicable in $V$, written as $\text{applicable}[V](\phi_D)^B$, iff $
exists \bar{g} \in I(V)$ s.t. $\Gamma_{\bar{g}}(\phi_D) = \text{true}$.

The execution of an assignment $\Omega$ on $V$, denoted as $\text{exec}[V](\Omega)^B$, is a valuation from the set $\{V' \mid \exists \bar{g} \in I(V), \text{s.t. } \forall p. V'(p) = \Gamma_{\text{update}(\bar{g}, \Omega)}(p)\}$.

The B-model contains more states and runs than the ground model: indeed, when a variable gets the value of some abstract expression, any possible consistent valuation should be considered as an effect of the assignment. As a result, the transition leads to different states, some of which may be unreachable in a real execution. In our example, if the implementation of the reduce function is not defined, we should consider any possible outcome of the assignment, including the case where the amount is not changed. Hence, an infinite loop is possible, where the Approver rejects the request and the amount does not change, which does not appear in real execution.

As a consequence, the B-model is not applicable for the verification of existential properties. If a run satisfying the property is contained in the B-model, it is not guaranteed to appear in the ground one. On the contrary, if the verification of an assertion is considered, the satisfaction of this property in the B-model implies also the satisfaction in the ground model.

**Definition 6.9 (Satisfiability in B-Model)**

The possibility $\phi$ is satisfied in B-model, written as $\Sigma^B \models_P \phi$, iff there
exists a run \( \omega^B \) of \( \Sigma^B \), s.t. \( \omega^B \models [\phi] \).

The assertion \( \phi \) is satisfied in B-model, written as \( \Sigma^B \models_A \phi \), iff for each \( \omega^B \) of \( \Sigma^B \), \( \omega^B \models [\phi] \).

**Theorem 6.2**

Given an assertion \( \phi \), if \( \Sigma^B \models_A \phi \), then \( \Sigma \models_A \phi \),

Given a possibility \( \phi \), if \( \Sigma^B \not\models_P \phi \), then \( \Sigma \not\models_P \phi \).

**Proof.** Assertion. From the definition of B-model interpretation, applicability, and execution follows, that for any run \( \omega \) of \( \Sigma \) there exists a corresponding run \( \omega^B \) of \( \Sigma^B \), where the same sequence of actions is performed, and for corresponding observable states \( \sigma_i = (\bar{s}, \bar{g}, \bar{u}) \) and \( \sigma_i^B = (\bar{s}, V, \bar{u}) \), \( i \geq 0 \), holds that \( \bar{g} \in I(V) \). Obviously, if \( \omega^B \models [\phi] \) then \( \omega \models \phi \). Since \( \omega^B \models [\phi] \) for any \( \omega^B \in \Xi(\Sigma^B) \), then the assertion is satisfied also in \( \Sigma \).

Possibility. Note that \( \Sigma^B \not\models_P \phi \iff \Sigma^B \models \neg \phi \). Therefore, \( \Sigma \models \neg \phi \), or in other terms \( \Sigma \not\models_P \phi \). □

### 6.2.5 Relations between Abstraction Models

The following theorem summarizes the relations between the satisfaction of the property under the two abstraction models.

**Theorem 6.3** Given an assertion \( \phi \),

\[
\Sigma^B \models_A \phi \implies \Sigma \models_A \phi \implies \Sigma^K \models_A \phi.
\]

Given a possibility \( \phi \),

\[
\Sigma^K \models_P \phi \implies \Sigma \models_P \phi \implies \Sigma^B \models_P \phi.
\]

**Proof.** Follows immediately from Theorem 6.1 and Theorem 6.2. □

The theorem suggests that the interplay of the two models of abstraction is used in order to put bounds on the satisfiability of the property.
6.3. REPRESENTATION OF DATA ASSUMPTIONS

While these models can not provide an exact answer for the property verification under an arbitrary composition model (the problem is undecidable in general), they are still able to return safe answers for such a problem. Moreover, they are able to provide exact answers for a wide range of the infinite state space systems.

This theorem also suggests the following verification algorithm. Given an assertion property, we check it under B-model. If the property is satisfied, it is satisfied also under the ground model, and true is returned. If the property is violated by B-model, we check if it is also violated by the K-model. If it is the case, then it is also violated in the ground model, return false. Otherwise, the satisfiability is not defined and further refinement is needed.

In the case of a possibility, we first check it under K-model. If it is satisfied, it is indeed satisfied by the ground model and true is returned. Otherwise, we check its violation under the B-model. If the possibility is also violated by this model, return false. Again, if neither of these holds, the satisfiability is not defined and we have to refine the model.

6.3 Representation of Data Assumptions

Due to intrinsic information incompleteness of the Web service specification, it is often difficult to analyze Web service compositions, if at all possible. Internal implementation details of service operations and decisions are often hidden, making the analysis results incomplete or even incorrect. In order to improve the outcome of the verification and validation procedure it is necessary to make certain assumptions on these details, under which the analyzed system is supposed to work properly. These assumptions may express the expected relations between the input and output of the service operations, potential values of the data exchanged between the
participants, postconditions on the service invocations, etc.

The assumptions may have two forms. First, it may be used to specify the constraints that the implementation of the service function must satisfy. This kind of constraints allows one to describe the relations between the input parameters and the output value. In our Loan Approval example the following constraint on the function \texttt{assess} may be defined:

\[(\text{small}(p)) \rightarrow (\text{result} = \text{"low"})\]

Here \(p\) denotes a formal parameter of the function, and \(\text{result}\) denotes the outcome of the function call. The assumptions of this form serve as invariants on the system execution.

Second, the assumptions may describe the values that the output of the service function may have. This kind of assumptions allows for modelling non-deterministic data operations that, nevertheless, take place in real executions (e.g., for modelling “opaque” assignments in BPEL). For instance, in order to model the fact that the Customer may request an approval for an arbitrary loan value, we can enumerate the following output valuations in the initial assignment (function \texttt{initial}):

\[\text{true} \rightarrow \text{large(result)} \sqcup \neg\text{large(result)} \sqcup \neg\text{small(result)} \sqcup \text{small(result)}\]

We use the operator \(\sqcup\) to distinguish the expressions that describe possible alternatives. Note, that the assumption of this kind may be used only in assignments, where there is a need to model non-deterministic outcome of the operation.

More formally, we represent the implementation assumptions as follows. We denote the properties of the first class as \(\mathcal{P}_{\text{Inv}}\) and the properties of the second class as \(\mathcal{P}_{\text{Alt}}\).

\textbf{Definition 6.10 (Implementation Assumptions)} Let \(f(x_1, \ldots, x_n)\) be a function. An invariant assumption in \(\mathcal{P}_{\text{Inv}}\) over the function has the form
6.3. REPRESENTATION OF DATA ASSUMPTIONS

\[ \text{precond-expr} \to \text{effect-expr}, \]

where precond-expr is an expression over input parameters, and effect-expr is an expression over input parameters/result value of the function. An alternation assumption in \( \mathcal{P}_{\text{Alt}} \) over the function has the form

\[ \text{precond-expr} \to \text{alternative} (\sqcup \text{alternative})^*, \]

where precond-expr is an expression over input parameters, and alternative is an expression over input parameters/result value of the function.

Given an assumption, we call its instantiation an expression, where the formal parameters are replaced with actual values.

**Example 6.1** The following property specifies an invariant assumption on the function \texttt{assess}:

\[(\text{small}(p)) \to (\text{result} = \text{"low"})\]

The assumption specifies that whenever the parameter is sufficiently small, the result of the function is “low”. Consider the following assignment in the Assessor process:

\[\text{response.risk} = \text{assess(request.amount)}\]

The assumption is then instantiated as the following invariant:

\[(\text{small(request.amount)}) \to (\text{response.risk} = \text{"low"})\]

At design-time the data constraints modelled in this way allow for refining the composition specification, representing those properties of the composition that are crucial for the correct implementation. At run-time these constraints may be further exploited as the properties to be monitored when the composition is executed.
6.4 Construction of the Abstraction

In this section we present the translation procedure that allows us to build an abstract global transition system from the specification of the composition. It consists of the definition of the representation of the functions applicable and exec for the given transition, constraints on the consistency of the valuation, interpretation of transition conditions, etc. The procedure is different for the two models of abstraction defined above.

When the translation procedures are defined, an abstract composition is obtained from the specification as follows.

• Select an appropriate communication model (e.g., using the analysis techniques presented in Chapter 5);

• Select a (sufficiently large) set of abstract propositions \( \mathcal{P}^A \);

• Constrain the initial valuations of the system;

• Construct the abstract transition relation using the translation procedures for transition applicability and effect.

In the abstraction the states are represented by the valuation of abstract propositions. Each ground state in the interpretation of the valuation should satisfy certain axioms implied by the model. These axioms include equality transitivity axioms, propositional consistency axioms, functional consistency axioms, implementation assumptions axioms, and certain auxiliary axioms (e.g., for certain predicates, constants, etc.).

Definition 6.11 (Consistency axioms)

*Every global ground state \( \bar{g} \) should satisfy all the axioms of the following forms:*

• Logical consistency:
\[\bar{g} \models \phi_1 \lor \bar{g} \models \phi_2 \quad \text{then} \quad \bar{g} \models \phi_1 \lor \phi_2;\]

\[\bar{g} \models \phi_1 \text{ and } \bar{g} \models \phi_2 \quad \text{iff} \quad \bar{g} \models \phi_1 \land \phi_2;\]

- **Equality transitivity:**

\[\begin{align*}
&\quad \text{if } \bar{g} \models (t_1 = t_2) \land \bar{g} \models (t_2 = t_3) \quad \text{then} \quad \bar{g} \models (t_1 = t_3); \\
&\quad \text{if } \bar{g} \models (t_1 = t_2) \land \bar{g} \models (t_2 \neq t_3) \quad \text{then} \quad \bar{g} \models (t_1 \neq t_3).
\end{align*}\]

- **Functional consistency (for the functions with fixed interpretation):**

\[\begin{align*}
&\quad \text{if } \bar{g} \models \bigwedge_{i=1}^{n} (t_{1i} = t_{2i}) \quad \text{then} \quad \bar{g} \models (f(t_{11}, \ldots, t_{1n}) = f(t_{21}, \ldots, t_{2n})).
\end{align*}\]

- **Assumption invariants:**

\[\begin{align*}
&\quad \text{given an assumption } (pe \rightarrow ee) \in \mathcal{P}_{\text{Inv}} \text{ and its instantiation } \\
&\quad (pe' \rightarrow ee'), \text{ if } \bar{g} \models pe' \text{ then } \bar{g} \models ee'.
\end{align*}\]

We remark, that the above list of rules may be extended if the abstraction is enhanced with additional constructs, like e.g., constants, enumerations, etc. In this case additional rules should be specified and added to the above list.

### 6.4.1 K-model Construction

The K-model of abstraction aims at representing a “pessimistic” view on a system. That is, the valuation of a proposition is true only if it is “known” to be true in all the corresponding ground states. On the other hand, one would like to have the maximal knowledge about the system, i.e., to avoid situations, where a proposition is mapped to false (is unknown), while the corresponding formula evaluates to true in all the ground states defined by the valuation. Consider, for instance, a valuation \{true, false\} over the propositions \{\(p_1, p_1 \lor p_2\}\), such that \(V(p_1) = \text{true}\) and \(V(p_1 \lor p_2) = \text{false}\). By definition the valuation is consistent, while \(\forall \bar{g} \in \mathcal{I}(V), \text{ if } \bar{g} \models p_1 \text{ then } \bar{g} \models p_1 \lor p_2\).
Therefore, the valuation of the proposition \( p_1 \lor p_2 \) may be set to true without affecting the interpretations. Since the evaluation of the formula is based on the valuation of abstract propositions, the latter should contain the “maximal knowledge” on these propositions. Such a valuation is referred to as deductive closure.

Given an arbitrary valuation \( V \), its deductive closure \( V^* \) is obtained by iteratively applying consistency axioms to the valuation. In general, the axioms may potentially be transformed to a set of rules of the following form: \( \psi \rightarrow \phi \), where \( \psi \) denotes the precondition of the rule and \( \phi = \bigwedge p^\text{post}_j \) is its consequence.

**Definition 6.12** Given a valuation \( V \), its deductive closure \( V^* \) is a transitive closure over the propositions in \( \mathcal{P}^A \) obtained by repeatedly applying each rule \( \psi \rightarrow \bigwedge p^\text{post}_j \) on the valuation:

\[
V \models \psi \implies \forall p^\text{post}_j, \ V^*(p^\text{post}_j) = \text{true}.
\]

Indeed, this construction terminates, and the following property holds for the obtained closure.

**Proposition 6.6** If \( V^* \models \phi \) then \( \forall \vec{g} \in \mathcal{I}(V), \ \vec{g} \models \phi \).

**Valuation consistency**

By construction, all the ground states in the interpretation of the deductive closure satisfy the consistency axioms. According to Proposition 6.2, the consistency requirement to be checked on the valuation is that it does not contain contradictory facts.

**Definition 6.13 (K-consistency)**

The valuation \( V \) is K-consistent, written as \( \text{consistent}(V)^K \), if \( \forall p, \lnot p \in \mathcal{P}^A \), if \( V^*(p) = \text{true} \) then \( V^*(\lnot p) = \text{false} \).
Applicability

We transform the applicability of the transition as follows. Since the transition condition may contain propositions not in $P^A$, according to the definition of the applicability under the K-model states that the transition is enabled only if the condition is “known” to be true. This knowledge has to be obtained given only the propositions in $P^A$. Given a formula $\phi$, we call its $K$-interpretation $\phi^K$ a formula defined as follows:

- obtain $\phi' = \bigvee \bigwedge l_{ij}$, such that $\phi' \Leftrightarrow \phi$;
- replace each $l_{ij}$ in $\phi$ with $\begin{cases} p, \text{ if } l_{ij} \in P^A \\ \text{false}, \text{ otherwise} \end{cases}$

Note that if $V \models \phi^K$ then $\forall \bar{g} \in \mathcal{I}(V), \bar{g} \models \phi$.

A transition with the data condition $\phi_D$ is applicable under the valuation $V$ if the valuation satisfies the K-interpretation of $\phi^K_D$.

**Proposition 6.7** If $V \models \phi^K_D$ then $\text{exec}[V](\phi_D)^K$.

**Proof.** Follows immediately from the definition of applicability. $\square$

Execution

The translation of the transition assignment is performed as follows. When an assignment is performed, the propositions over the variable, to which the value is assigned, become unknown. The proposition that relates the variable with a new value becomes true.

Let us denote the abstract propositions, where a variable $x$ appears, as $P^A(x)$.

**Proposition 6.8** The execution $\text{exec}[V](\Omega)^K$ of an assignment $\Omega = \{x_i := t_i\}$ on the valuation $V$ is constructed as follows:

1. $V_0 := V$;
2. for each assignment $x_i := t_i \in \Omega$

- $V_i := V_{i-1}$;
- $\forall \ p \in \mathcal{P}^A(x_i), V_i(p) := false$;
- $V_i(x_i = t_i) = true$;

3. $\text{exec}[V](\Omega) := V^*_n$.

**Proof.** Immediately follows from the definition of the transition effect under the K-model and the definition of the update of the ground state. □

When a variable is assigned a value of the function for which an alternation assumption is defined, each alternative potentially leads to a separate valuation. Such a valuation is defined both by the transition assignment and the assumption definition. Let as denote an expression $\phi$ where the occurrences of the term $t$ are replaced with the variable $x$ as $\phi[t/x]$.

**Definition 6.14 (Alternation assignment under K-model)**

If for a function $f$ an alternation assumption $pe \rightarrow alt_1 \sqcup \cdots \sqcup alt_n$ is defined, then for any transition with an assignment $\Omega \cup \{x := f(t_1, \ldots, t_n)\}$ the executions on the valuation $V$ are defined as follows:

1. $V' = \text{exec}[V](\Omega)$;

2. if $V' \models pe$ then for some $alt_i$ apply the rule $pe \rightarrow alt_i[f(t_1, \ldots, t_n)/x]$, otherwise for every $p \in \mathcal{P}^A(x), V'(p) = false$;

3. $\text{exec}[V](\Omega \cup \{x := f(t_1, \ldots, t_n)\}) = V'^*$.

In other words, when the precondition of the assumption is satisfied, the target valuation represents one of the possible alternatives as specified in the assumption definition. If the precondition is not satisfied, the value of the variable is unknown.
Consider, for instance, the execution of the assignment \( \text{amount} := \text{initial}() \), where the assumption \( \text{true} \rightarrow \text{large(result)} \sqcup \neg \text{large(result)} \) is defined for the function \( \text{initial} \), on the valuation \( V(\text{large(result)}) = \text{false} \), \( V(\neg \text{large(result)}) = \text{false} \). The effect is defined by the following two alternative valuations: \( V_1 = \{ \text{true}, \text{false} \} \) and \( V_2 = \{ \text{false}, \text{true} \} \).

The initial state of the abstraction under the K-model is defined in such a way that no facts are “known”: \( V_0 = V''^* \), where for each \( p \in \mathcal{P}^A \), \( V'(p) = \text{false} \).

### 6.4.2 B-model Construction

#### Valuation consistency

According to the definition of the B-model, there is no need to create another valuation from the given one, as it is done in case of the K-model. The consistency of the valuation may be checked by verifying all the consistency axioms and the satisfaction of Proposition 6.4.

**Definition 6.15 (B-consistency)** The valuation \( V \) is B-consistent, written as consistent \( (V)^B \), if it satisfies all the defined consistency axioms, and if \( \forall \ p, \neg p \in \mathcal{P}^A, V(p) = \text{true} \Leftrightarrow V(\neg p) = \text{false} \).

#### Applicability

The B-model represents an “optimistic” view on the composition. That is, all the unclear facts are interpreted as possible. Given a formula \( \phi \), we call its B-interpretation \( \phi^B \) a formula defined as follows:

- obtain \( \phi' = \bigvee \bigwedge l_{ij} \), such that \( \phi' \Leftrightarrow \phi \);
- replace each \( l_{ij} \) in \( \phi \) with \[
\begin{cases}
p, & \text{if } l_{ij} \in \mathcal{P}^A \\
\text{true}, & \text{otherwise}
\end{cases}
\]
Note that if $V \not\models \phi^B$ then $\forall \bar{g} \in \mathcal{I}(V), \bar{g} \not\models \phi$.

The transition is applicable, when the B-interpretation of the condition evaluates to true under the valuation.

**Proposition 6.9** If $V \models \phi^B_D$ then applicable$^B[V](\phi_D)^B$.

**Proof.** Follows immediately from the definition of the applicability under the B-model. □

**Execution**

The effect of the transition is any consistent valuation that may be obtained from the source valuation as follows.

**Proposition 6.10** The execution $V' = \text{exec}^B[V](\Omega)$ of the assignment $\Omega = \{x_i := t_i\}$ on the valuation $V$ is an arbitrary valuation, such that

- consistent$(V')^B$, and
- $V'(p) = \begin{cases} \{\text{true}, \text{false}\}, & \text{if for some } i, \ p \in \mathcal{P}^A(x_i) \\ \text{true}, & \text{if for some } i, \ p \equiv (x_i = t_i) \\ V(p), & \text{otherwise} \end{cases}$

**Proof.** Immediately follows from the definition of the transition effect under the K-model and the definition of the update of the ground state. □

In case of the B-model of abstraction, the alternation assumption allows to restrict all the possible target valuations only to those allowed by the assumption definition.

**Definition 6.16 (Alternation assignment under B-model)**

*If for a function $f$ an alternating assumption $pe \rightarrow alt_1 \sqcup \cdots \sqcup alt_m$ is defined, then for any transition with an assignment $\Omega \cup (x := f(t_1, \ldots, t_n))$ the executions on the valuation $V$ is an arbitrary valuation $V'$ such that*
• consistent($V'$)$_B$, and
• $V'(p) = \begin{cases} 
\{true, false\}, & \text{if } p \in \mathcal{P}^A(x) \\
\text{exec}[V](\Omega)(p), & \text{otherwise}
\end{cases}$
• if $V \models pe$ then $V' \models \bigvee_{i=1}^m alt_i[f(t_1, \ldots, t_n)/x]$

The initial states of the abstraction under the B-model are defined by a set of arbitrary consistent valuations: $V_0 \in \{V' \mid \text{consistent}(V')^B\}$.

### 6.5 Discussion

We presented an approach that extends the analysis of Web service compositions with the capability to take into account data flow properties. The approach is based on the abstraction techniques representing properties in the logic with equalities and uninterpreted functions. This model is generic enough and allows for representation of infinite data domains, arbitrary functions and predicates, while preserving efficiency in reasoning.

We presented two models of abstraction that allow us to define the bounds for the property satisfiability, and show how the analysis of a real system may be performed using the verification of these abstractions. Whenever the result cannot be immediately obtained due to a lack of information, the explicit data flow constraints may be incrementally added to the specification in order to refine the model until a precise answer is obtained. These constraints represent certain assumptions over the semantics of unknown or hidden service implementations, and may be further controlled (monitored) at run-time.

It is easy to see that the encoding of K-model is considerably more complex that the encoding of B-model. This problem may be easily resolved as follows. Any property is first analyzed in B-model of the composition. The obtained trace (assertion violation or possibility witness) if appears should
also appear in K-model. Therefore, it is suffice to build a K-representation of the trace instead of the full model, which is much simpler task.

It is worth to be noticed that the presented approach may be efficiently implemented using symbolic model checking techniques. Indeed, the main problem of the analysis task here is representation and management of propositional formulas, which is the main concerns of such techniques. This also advocates our choice of NuSMV model checker as an implementation platform: it is particularly suited for the systems defined symbolically, i.e., using propositional logic for transitions rules, invariants, axioms.

As yet, the problem of management data gained a few attention in the Web service analysis. In [57] the modelling of structured XML data and operations, their further analysis are addressed. The data domains, however, made finite and non-determinism is omitted, leading to intrinsically incomplete results. Moreover, the computational cost growth exponentially when the domain increase, while in our approach this problem is irrelevant.

Our work is close to other works on data abstractions, in particular to counterexample-guided abstraction refinement [32], and 3-valued verification [22]. Differently to these approaches, we focus on resolving information incompleteness, rather then finding a small set of propositions for efficient verification of systems with known semantics.

In the presented approach the specification is refined by a set of data flow assumptions that remove the information incompleteness. These assumptions, however, should be defined manually by the analysts, or extracted from the domain knowledge. A much more interesting problem is to find these assumptions automatically, by the analysis of the produced results, e.g., counterexample traces. The analysis problem then changes: instead of the verification of a property, the tool automatically extract constraints, under which the property is satisfied/violated. This kind of analysis is currently an ongoing work.
Chapter 7

Modelling and Analysis of Time Properties

In this chapter we present an approach that extends our framework with the ability to represent and analyze time-related requirements over Web service compositions. Timed properties not only change the landscape of the non-functional service characteristics, but may also affect the behavior of the composition, changing the qualitative properties of the system. This problem is especially important in the context of long-running business processes, where the ability to deliver required functionality depends on the timing of the business activities and their relations, while the transactional control is performed through the mechanism of timeouts.

We present formal notations for expressing timed requirements in the composition specification. The requirements are provided in two forms: basic properties that express the constraints on activity durations, and complex properties that define structure and timing characterization of behavioral intervals. We also present techniques for the verification and computation of timed requirements. While the verification allows for checking the satisfiability of a (timed or non-timed) property, the presented symbolic computation algorithms are used to find extremal (i.e., minimal/maximal) time bounds, where the property holds.
7.1 Case Study: e-Government Application

We illustrate our approach with an e-government application. The goal of the application is to provide a service that manages user requests to open sites for the disposal of dangerous waste. According to the existing Italian laws, such a request involves the interaction of different actors of the public administration, namely a Citizen Service, a Waste Management Office (WMO), a Secretary Service, a Procedure Manager, a Technical Committee, and a Political Board. In this application, the whole procedure is implemented as a composition of Web services that serve as interfaces to the processes of the above actors. We model the composition using BPEL specifications to describe the interactions among these actors. The high-level model of the composition is presented in Fig. 7.1. The procedure describes different phases of the application management where the request is registered, the documentation is evaluated and collected, the application is analyzed regarding the ecological impact of the site, the public conference is scheduled and organized, and final decision is provided.

Apart from the functional requirements, the execution of the process in the choreography should respect a set of timed requirements and constraints, dictated by Italian laws or by the agreement among the involved parties. These requirements (callouts in Fig. 7.1) specify, for example, that the period of time between the application registration and the notification of the Procedure Manager should not exceed 4 weeks, or that the participants can change the date within 1 week after the preliminary call. The behavior of the composition and the possibility to satisfy these requirements depend on the time needed for the execution of the activities the involved parties are responsible for. We remark that the critical parameter is the duration of internal activities of the participants, and not the communication time, which can be neglected.
In these settings, the composition analysis may become a long, error-prone process of finding boundary time values that would ensure correctness of the composition with respect to functional and timed requirements. Consider, for instance, the problem of determining the maximum time to be spent by the Citizen to provide integration documents and by the Technical Committee to perform the analysis, such that the requirement to announce a conference within 4 weeks after registration is satisfied.

The analysis of time-related aspects of the compositions requires explicit representation of timeouts, operation durations, and even complex properties expressing various timed requirements. While timeouts can be...
represented in BPEL, durations and timed requirements can not, and require specific way to be modelled. In our framework, we assume that the answer times are negligible by default, and that activities that have a non-negligible duration are annotated in the BPEL specification with an extra *duration* attribute. In Fig. 7.1 an excerpt of the annotated BPEL is presented. Here a BPEL event handler “date modification” is used to model a time-bounded possibility to change the date of the conference. That is, the *onAlarm* activity is triggered if the user does not call the “modifyDate” operation within 1 week. On the contrary, the internal activity “verify reviews” is equipped with a duration annotation to express that certain time may be used for the reviews analysis.

The representation of *timed requirements* requires more powerful notations. Consider, for instance, the requirement that the interval from the registration to the conference call should not exceed 4 weeks, and it is followed by the interval of length of at least 2 weeks, ending with the conference. This requirement spans over many activities performed by different parties. In order to be able to handle it, it is necessary to provide a model of the behavior of the BPEL processes that allows for an explicit representation of time. Moreover, it is necessary to exploit techniques for reasoning about time to check if this requirement is satisfied by the BPEL timed model. In the following sections we demonstrate how these issues can be addressed with the help of the STS composition formalism, the duration annotations and duration calculus for complex time requirements.

### 7.2 Modelling Timed Requirements

The timed properties of the analyzed BPEL composition may have two forms. Simple properties, or BPEL *duration annotations*, allow to specify the possible duration of a particular BPEL activity (either basic or
CHAPTER 7. MODELLING AND ANALYSIS OF TIME PROPERTIES

complex). Complex properties, on the other hand, provide a way to express requirements on the time intervals, their structure and quantitative characteristics.

7.2.1 BPEL Duration Annotations

BPEL duration annotations allow for representing simple timed assumptions on the process execution, like service response time, duration of some internal operation or sequence of operations, etc. In this case the activity is explicitly annotated with the special constraint. This annotation is used in our case study, for example, to denote that the duration of the “verify activity” is less than or equal to 1 week (Fig. 7.1). Such constraints are conjunctions of the clauses of the form \( \text{dur} \sim c \), where \( \sim \in \{<, >, \leq, \geq, =\} \).

Table 7.1 represent the translation of these constructs. When applied to an atomic activity (e.g. invoke, empty), this annotation is semantically equal to the sequence of two transitions. A fresh clock variable \( x_t \) is defined for this activity. The first transition, from \( s_b \) to \( s' \), is an instant transition and resets the clock \( x_t \). The second transition, from \( s' \) to \( s_e \) has the guard that evaluates to true if the value of the clock \( x_t \) satisfies the duration constraints. An invariant is defined in the intermediate location. The rule for assigning invariants is the following:

- if \( \phi_T = (x_t \leq c) \) then \( \mathcal{L}_I(s_1) = (x_t \leq c) \);
- if \( \phi_T = (x_t < c) \) then \( \mathcal{L}_I(s_1) = (x_t < c) \);
- if \( \phi_T = (x_t \geq c) \) or \( \phi_T = (x_t > c) \) then \( \mathcal{L}_I(s_1) = \text{true} \);
- if \( \phi_T = (x_t = c) \) then \( \mathcal{L}_I(s_1) = (x_t \leq c) \).

When the duration annotation is specified for a structured activity, the translation is more complex. First, all instant subactivities should be converted to non-instant. That is, time can pass arbitrary with these activities.
Table 7.1: Mapping duration annotations to STS

<table>
<thead>
<tr>
<th>Activity Example</th>
<th>STS representation</th>
</tr>
</thead>
</table>
| basic-activity a duration="lessEqual(3)" | \((s_b, true, \tau, 0, \{x_t\}, true, s') \in \mathcal{R}\)  
  \((s', x_t \leq 3, a, 0, 0, false, s_e) \in \mathcal{R}\)  
  \(L_I(s') \equiv x_t \leq 3, s_b, s', s_e \in \mathcal{S}\) |
| complex-activity a duration="lessEqual(3)" | \(s_b \rightarrow s_e\)  
  \(\forall (s', \phi, a', \Omega, Y, u, s'') \in \mathcal{R}, \text{ if } s_b \rightarrow^* s' \text{ and } s'' \rightarrow^* s_e\)  
  then \(u = false\)  
  \(\Sigma'_\Delta = \Sigma_\Delta \times A([s_b]^0 \sim true \sim [s_e]^0 \rightarrow len \leq 3)\) |

Second, the specification should be constrained to include only the behaviors, in which this complex activity satisfies the duration requirement. This is done by adding to the specification a constraint stating that the time interval from the beginning of the activity \(a\) (location \(s_b\)) to its end \((s_e)\) has the required duration. A corresponding automaton is generated for the constraint, and the synchronous (i.e., lock step) product of the GTS \(\Sigma_\Delta\) and the automaton is built. This constraint is specified in the duration calculus, which we will describe in the next section.

**Example 7.1** An excerpt of the STS representing the BPEL code in Fig. 7.1 is illustrated in Fig. 7.2\(^1\). Here the activity “verify reviews”, equipped with the duration annotation, is modelled as a sequence of two transitions. The first transition is instant, while the second can be fired within 3 time units. After this activity, the conference call is instantaneously sent to the partners, and a “modifyDate” message is awaited. If the message is received within 5 time units, then a customer-defined date is immediately verified. If the message is not received within 5 time units, the transition corresponding to the \(onAlarm\) activity is fired, and a final conference date is calculated. In both cases the process instantaneously sends a final confer-

--

\(^1\)For the sake of simplicity we omit the data-related constructs. We also represent non-urgent transitions as double lines, output actions as !op, and input actions as ?op.
ence call to the partners.

### 7.2.2 Complex Timed Requirements

We now present a language for specifying complex timed requirements to be verified on the global specification of the composition. These requirements are used to represent certain complex timed assumptions that are hard to state as timeouts or as constraints on activity duration. Such re-
MODELLING TIMED REQUIREMENTS

Requirements may express the time intervals between events (or a sequences of events), time bounds on some condition to hold or even complex logical combinations on them. In order to express complex timed requirements we exploit a subset of duration calculus (DC) [28]. It allows us to express properties of finite sequences of behaviors and to measure the duration of a given behavioral fragment. In particular, it is possible to express the time intervals between events (or a sequences of events), time bounds on some condition to hold and even complex logical combinations on them.

More formally, the logic is defined as follows. DC formulas are evaluated over finite behaviors, i.e., over finite sequences of observable states and actions $\omega = \sigma_b, a_b, \ldots, \sigma_e, a_e$. Let us denote the number of steps in the sequence $\omega$ as $\#\omega$. Let also $dom(\omega) = \{0, 1, \ldots, \#\omega - 1\}$ be the set of positions in $\omega$.

The set of intervals of $\omega$ is given by $Intv(\omega) = \{[b, e] \in dom^2(\omega) \mid b \leq e\}$, where the interval identifies a subsequence of $\omega$ between positions $b$ and $e$.

Let $p$ range over propositions formula, that is propositions over states, actions, variables, and their boolean combinations; $d, d_1, d_2$ over DC formulas, $c$ range over natural number constants, and $\sim \in \{<, \leq, =, >, \geq\}$. The DC formulas are defined as follows:

\[
d := [p]^0 \mid [\lceil p \rceil] \mid d_1 \sim d_2 \mid d_1 \land d_2 \mid \neg d \mid len \sim c \mid len(p) \sim c
\]

Other boolean operators over DC formula may be added in a usual way.

Intuitively, $[p]^0$ means that the interval is a point and at that point the proposition $p$ holds. $[\lceil p \rceil]$ requires the propositions to hold at each position of the interval, while $d_1 \sim d_2$ states that the interval consists of two subintervals, first satisfying $d_1$ and the second satisfying $d_2$. Formula $len \sim c$ requires that the time length of the interval is $\sim$ the constant $c$, while the formula $len(p) \sim c$ states that the time where $p$ holds is $\sim c$ for the interval.
We denote the satisfaction of DC formula \( d \) on the finite sequence \( \omega \) as \( \omega \models d \). We write also \( \omega, i \models p \) to denote that \( \sigma_i \) and/or action \( a_i \) satisfy \( p \). The satisfaction of the formula on the sequence \( \omega \) and the interval \([b, e] \in \text{Intv}(\omega)\), denoted as \( \omega, [b, e] \models d \), is inductively defined as follows:

\[
\begin{align*}
\omega, [b, e] \models [p]^0 & \iff b = e \text{ and } \omega, b \models p; \\
\omega, [b, e] \models [p] & \iff b < e \text{ and for all } b \leq i < e, \omega, i \models p; \\
\omega, [b, e] \models d_1 \diamond d_2 & \iff \text{ for some } i, \ b \leq i \leq e, \ \omega, [b, i] \models d_1 \text{ and } \omega, [i, e] \models d_2; \\
\omega, [b, e] \models d_1 \land d_2 & \iff \omega, [b, e] \models d_1 \text{ and } \omega, [b, e] \models d_2; \\
\omega, [b, e] \models \neg d & \iff \omega, [b, e] \not\models d; \\
\omega, [b, e] \models \text{len} \sim c & \iff (t_e - t_b) \sim c; \\
\omega, [b, e] \models \text{len}(p) \sim c & \iff \sum_{i=b}^{e-1} \begin{cases} 
  t_{i+1} - t_i & \text{if } \omega, i \models p \\
  0 & \text{otherwise}
\end{cases} \sim c.
\end{align*}
\]

Additionally, we define the following derived constructs:

- \([p]\) \(\overset{\text{def}}{=} ([p] \sim [p]^0)\) states that the proposition holds over the interval including the endpoint;
- \(\diamond d \overset{\text{def}}{=} \text{true} \sim d \sim \text{true}\) requires \( d \) to hold for some subinterval of the behavior;
- \(\square d \overset{\text{def}}{=} \neg \diamond \neg d\) denotes, that \( d \) holds for all subintervals.

Finally, the satisfaction of the DC formula on the sequence \( \omega \) is defined as follows:

\[\omega \models d \iff \omega, [0, \#\omega - 1] \models d.\]

**Example 7.2** Let us consider the requirement represented in Fig. 7.1 stating that the interval from the protocol registration to the conference call should not exceed 4 weeks, and that from the call to the conference at least
2 weeks should pass. The requirement may be graphically represented as shown in Fig. 7.3.

This requirement may be expressed with the following DC formula:
\[ \Box([\text{registration}]^0 \land true \land [\text{conference}]^0 \rightarrow (len \leq 4) \land [\text{sendCall2Cust}]^0 \Rightarrow (len \geq 2)) \]

The formula says that, for all intervals of the behavior, if the registration happens at the beginning of the interval and the conference at the end, then the interval consists of two intervals with the call in between, such that the duration of the first is less than 4 weeks, and the duration of the second does not exceed 2 weeks.

### 7.3 Analysis of Time-related Properties

We now present a set of techniques and algorithms that can be used for the automated analysis of the qualitative and quantitative properties of Web service compositions. In particular, we show how the discrete model of time may be used for this purposes. We demonstrate how the various timed properties may be used through the analysis process, and present symbolic techniques for the calculation of time bounds, where the given property holds.

The following components are used in different phases of the analysis process:

- composition specification \( \Sigma_\Delta \) that represents a composition of BPEL processes, possibly annotated with the duration constructs;
set of complex timed constraints $\mathcal{P}_C^T$ expressed as DC formulas;

- set of target properties $\mathcal{P}_T^T$ to be verified against the composition;

- set of properties $\mathcal{P}_V^T$ to be measured using the quantitative algorithms.

### 7.3.1 Discrete Time Representation

In the implementation of the presented analysis techniques we adopt discrete model of time. The semantics of the duration calculus in this case may be given by the *Quantified Discrete-time Duration Calculus* (QDDC, [113]). The analysis based on the dense model of time under certain restrictions may be implemented in a similar way. See [114] for details.

**Discrete Model of GTS**

The *discretization* of the GTS formalism is performed as follows.

- Every clock variable is represented as an integer variable. The upper bound for the timer is set to the maximal constant appearing in the specification. After reaching the bound the value of the timer is not changed later. The results of [4] ensure the correctness of such a bound with respect to the behavior of the system.

- The transition where time passes synchronously increment values of these variables by one: $\langle \bar{s}, \bar{g}, C, \bar{u} \rangle \xrightarrow{\text{tick}} \langle \bar{s}, \bar{g}, C, \bar{u} + 1 \rangle$.

- The time passing event is represented with the special boolean variable *tick* that evaluates to true if and only if the time passing transition happens.

**Discrete-time Duration Calculus**

Under the assumption of discrete time model we use a subset of QDDC logic [113]. In this logic, however, the semantics of measurement operators
(i.e., \(\text{len}\) and \(\text{len}(p)\)) is different: it counts number of positions instead of
time durations. In order to correctly manage the measurements in QDDC,
we use the fact that the time passing event is marked with the \(\text{tick}\) boolean
variable. The operators are defined as follows:

\[
\omega, [b, e] \models \text{len} \sim c \quad \text{iff} \quad \sum_{i=b}^{e-1} \left( \begin{array}{cc} 1 & \text{if } \omega, i \models \text{tick} \\ 0 & \text{otherwise} \end{array} \right) \sim c. \\
\omega, [b, e] \models \text{len}(p) \sim c \quad \text{iff} \quad \sum_{i=b}^{e-1} \left( \begin{array}{cc} 1 & \text{if } \omega, i \models p \land \text{tick} \\ 0 & \text{otherwise} \end{array} \right) \sim c.
\]

When the composition is finitely represented with a set of atomic propo-
sitions (e.g., using the techniques presented in Chapter 6), the requirements
expressed as QDDC formulas may be effectively expressed and analyzed.
Let \(\mathcal{P}\) be a set of propositions, and \(V: \mathcal{P} \rightarrow \{\text{true}, \text{false}\}\) be a valuation
of these propositions. The following result defines the decidability of the
QDDC logic and the analysis algorithms.

**Theorem 7.1 ([113])** For every QDDC formula \(d\) we can effectively con-
struct a finite state automaton \(A(d)\) over the alphabet \(\mathcal{P}(d)\) such that for
all \(\omega, \omega \models d \iff \omega \in L(A(d))\), where \(L(A(d))\) is the language accepted by
the automaton \(A(d)\).

### 7.3.2 Usages of Timed Requirements in the Analysis

As we already mentioned, the complex timed requirements may be used in
the analysis of the composition in several ways.

**Constraining the Specification**

The complex time properties may express assumptions or constraints on
the system behavior. That is, in the analysis we should consider only those
behaviors that satisfy such a set of QDDC constraints $\mathcal{P}_C^T$. In order to represent this restriction, each constraint property $p_i \in \mathcal{P}_C^T$ is translated into the finite state automaton $A(p_i)$ that recognizes all and only the behaviors that satisfy $p_i$ (see Theorem 7.1). Finally, a synchronous (i.e. lock-step) product $(\Sigma \times A(p_1) \times \cdots \times A(p_n))$ of the specification and the properties automata is constructed. Indeed, this product describes the specification that satisfies all these constraints. This product is used for further analysis of the composition.

**Verification of Complex Requirements**

The composition specification $\Sigma$ can be directly verified against property $p_i \in \mathcal{P}_C$. This is performed as follows. First, a finite state automaton $A(\neg p_i)$ of the negation of the property is constructed. Second, a synchronous product $(\Sigma \times A(\neg p_i))$ of the specification and the automaton is built. If this product is not empty, then the behavior of the composition violates the property $p_i$.

**Computation of Time Bounds**

One can apply quantitative analysis to measure a certain property $p_i \in \mathcal{P}_V$. This analysis allows to extract time bounds in which the property holds, e.g. minimal time needed to complete the Waste Management application procedure, or an maximal time of the technical analysis that still allows to satisfy all the composition requirements. While the qualitative analysis of the above properties is widely studies in the literature, the computation of the timed properties requires new special techniques. In the following section we present the quantitative analysis algorithms and procedures.
7.4 Quantitative Analysis

While the DC logic provides a powerful formalism to specify complex quantitative timed properties of the Web service compositions, computing these properties requires finding by trial and error those values that make the property valid. Indeed, this technique is inherently incomplete.

Here we present an algorithm that given a composition specification and a certain property (in QDDC), computes the extremal (i.e. least/greatest) duration of the interval satisfying this property. The algorithm relies on previous results of [115], where the computation of extremal values for synchronous systems is addressed, and on the symbolic condition count algorithms of [23]. We denote the corresponding functions as \( MINDUR(d, \Sigma_\Delta) \) and \( MAXDUR(d, \Sigma_\Delta) \), where \( d \) is the computed QDDC formula and \( \Sigma_\Delta \) is a underlying composition model.

Intuitively, the algorithm performs as follows. First, the initial system \( \Sigma_\Delta \) and a property \( p \) are transformed into a system \( \Sigma'_\Delta(st, fin) \). Minimal (maximal) length of interval between a state satisfying formula \( st \) and a state satisfying formula \( fin \) in \( \Sigma'_\Delta \) is equivalent to the minimal (maximal) duration of interval satisfying \( p \) in \( \Sigma_\Delta \). Second, the actual extremal value is computed in \( \Sigma'_\Delta \) using the min/max interval functions \( \text{min\_interval}(st, fin, \Sigma'_\Delta) \) and \( \text{max\_interval}(st, fin, \Sigma'_\Delta) \). These functions compute the length of the interval from some state in \( st \) to a state in \( fin \).

7.4.1 Min/Max Interval Algorithms

The algorithms represented in Alg. 2 are used to calculate a minimum or maximum length of interval that starts in a state satisfying a \textit{start} condition, and finishes in a state satisfying an \textit{end} condition. More precisely, these algorithms compute the number of transitions, where the \textit{tick} variable is true, which is semantically equivalent to the time-passing transition.
Algorithm 2 Min/Max interval algorithms

1: procedure min_interval($start, final, \Sigma_\Delta$)
2: \[ R := start \cap \text{reachable}(final) \]
3: if $(R = \emptyset)$ return $\infty$
4: \[ min := 0 \]
5: \[ R := \text{immediate}(R) \cap \text{reachable}(final) \]
6: while $(R \cap final = \emptyset)$ do
7: \[ min := min + 1 \]
8: \[ R := \text{delayed}(R) \cap \text{reachable}(final) \]
9: \[ R := \text{immediate}(R) \cap \text{reachable}(final) \]
10: end while
11: return $min$
12: end procedure

13: procedure max_interval($start, final, \Sigma_\Delta$)
14: \[ R := start \cap \text{reachable}(final) \]
15: if $(R = \emptyset)$ return $0$
16: \[ max := 0 \]
17: \[ SR := \emptyset \]
18: \[ R := \text{immediate}(R) \cap \text{reachable}(final) \]
19: while $(R \notin SR)$ do
20: \[ SR := SR \cup \{R\} \]
21: \[ R := \text{delayed}(R) \cap \text{reachable}(final) \]
22: \[ R := \text{immediate}(R) \cap \text{reachable}(final) \]
23: if $R = \emptyset$ return $max$
24: else \[ max := max + 1 \]
25: end while
26: return $\infty$
27: end procedure

In these algorithms several function are exploited, namely \text{reachable}(R), \text{immediate}(R), and \text{delayed}(R). The first represents a set of states from which the set $R$ is reachable. The second represents a set of states that are reachable from any state in a set $R$ only through instant transitions. The third represents a set of states that are reachable from any state in $R$
through exactly one TICK transition. Note that in latter cases we restrict only to the states, from which the states satisfying final are reachable.

**Definition 7.1 (Reaching set)**
The set of states from which the states in \( R \) are reachable, denoted as \( \text{reachable}(R) \), is defined as

\[
\text{reachable}(R) = \{ \gamma | \exists \gamma_1 \in R, \ \exists \hat{\omega}, \ \text{s.t.} \ \gamma \xrightarrow{\hat{\omega}} \gamma_1 \}
\]

where \( \hat{\omega} = \gamma_1, a_1, \ldots, a_n, \gamma, n \geq 0 \).

**Definition 7.2 (Immediate set image)**
The set of states reachable from the given set \( R \) only through instant transitions, denoted as \( \text{immediate}(R) \), is defined as follows:

\[
\text{immediate}(R) = \{ \gamma | \exists \gamma_1 \in R, \ \exists \hat{\omega}, \ \text{s.t.} \ \gamma_1 \xrightarrow{\hat{\omega}} \gamma \},
\]

where \( \hat{\omega} = \gamma_1, a_1, \ldots, a_n, \gamma, n \geq 0, \ \text{and} \ \forall \ i \leq n, \ a_i \neq \text{TICK} \).

**Definition 7.3 (Delayed image)**
The set of states reachable from the given set \( R \) through exactly one TICK transition, denoted as \( \text{delayed}(R) \), is defined as follows:

\[
\text{delayed}(R) = \{ \gamma | \exists \gamma_1 \in R, \ \text{s.t.} \ \gamma_1 \xrightarrow{\text{TICK}} \gamma \}
\]

The minimal interval algorithm performs as follows. Initially, the set of states \( R \) is a set that satisfies start and from which the state satisfying final is reachable (\( \text{reachable}(\text{final}) \)). If the set is empty, there is no interval from start to final and the result of computation is infinity. Then we iteratively move from this set of states to their successors until the states satisfying final are reached. When the tick transition is performed, the value of min is incremented.

The maximal interval algorithm is implemented analogously. It progresses from the initial state trying to stay in the states not satisfying
final. If the is not possible, the current max value is returned. In order to detect an infinite cycle we calculate also a set of sets of states SR that contains all the images visited so far. If the cycle is detected (R ∈ SR), the algorithm returns infinity.

Let us define a finite subsequences of (potentially infinite) executions of the composition Σ_Δ as subexec(Σ_Δ). Let us also define the duration of sub-execution ω as len_ω: lenω def = \sum_{i=0}^{#ω-1} \begin{cases} 1 & \text{if } ω, i \models tick \\ 0 & \text{otherwise} \end{cases}.

**Theorem 7.2**

\[
\begin{align*}
\min\text{-interval}(start, final, Σ_Δ) &= \min \{len_ω \mid ω \in \text{subexec}(Σ_Δ), ω, 0 \models start, ω, #ω - 1 \models final\} \\
\max\text{-interval}(start, final, Σ_Δ) &= \max \{len_ω \mid ω \in \text{subexec}(Σ_Δ), ω, 0 \models start, ω, #ω - 1 \models final\}
\end{align*}
\]

**Proof.**

\textbf{min\text{-interval.} Let us define the shortest duration of the interval starting in a state satisfying start and ending at a state satisfying final as D. Let also S_{min} be a set of states that are reachable from a state in start at least in min time units, and from which the states in final are still reachable. The correctness of the algorithm follows from the following loop invariants:}

- \( min \leq D \);
- \( R = S_{min} \) (when the loop condition is evaluated).

Obviously, the invariants are satisfied before entering the loop. The invariant on \( R, R = S_{min}, \) and the loop test imply that \( min < D \). Therefore, \( min + 1 \leq D \) and the value of \( min \) may be incremented without affecting the invariant. The assignment of \( R \) to delayed implies that \( R \) will contain all the states reachable from current only through TICK transition, that is in one time unit. The next assignment looks for the states reachable
through zero or more immediate transitions, where time does not pass. Therefore, the invariant \( R = S_{\text{min}} \) is satisfied for the new value of \( \text{min} \).

The termination is guaranteed by restricting the working set to the states from which \( \text{final} \) is reachable. First, the set \( R \) is not empty (otherwise the states in \( \text{final} \) should be reached on previous steps). Second, since there is a path from \( \text{start} \) to \( \text{final} \) from each of the visited states, eventually \( \text{final} \) will appear in \( R \).

On the termination we have that \( R = S_{\text{min}} \), and \( \text{min} \leq D \). From the definition of \( S_{\text{min}} \) and the loop condition follows that \( D \leq \text{min} \). Therefore, \( D = \text{min} \).

\( \text{max}_\text{interval} \). As before, let us define the longest duration of the interval starting in a state satisfying \( \text{start} \) and ending at a state satisfying \( \text{final} \) as \( D \). Let also \( S_{\text{max}} \) be a set of states that are reachable from a state in \( \text{start} \) at least in \( \text{max} \) time units, from which the states in \( \text{final} \) are still reachable. The correctness of the algorithm follows from the following loop invariants:

- \( \text{max} \leq D \);
- \( R = S_{\text{max}} \) (when the loop condition is evaluated).

Obviously, the invariants are satisfied before entering the loop. The two assignments set the value of \( R \) to the set reachable from the previous through exactly one time unit. If the set \( R \) is empty then there is no \text{TICK} successors from \( S_{\text{max}} \) to \( S_{\text{max}+1} \) such that states in \( \text{final} \) can be reached. Therefore, no longer path from \( \text{start} \) to \( \text{final} \) can exist than that discovered on previous step, and \( D \leq \text{max} \). By the invariant we have \( \text{max} \leq D \) and thus \( \text{max} = D \). If the set \( R \) is not empty, we have that \( \text{max} < D \), and the increment does not violate the invariant.

To see that the algorithm terminates, consider the case of a cycle, where from every state the states in \( \text{final} \) are still reachable. This cycle includes
the tick transitions (otherwise \( R \) would be empty). To detect the cycle, we the set of sets of states \( SR \) is used. This set increases monotonically, and since the number of states is finite, eventually \( R \in SR \). Hence the loop test guarantees the termination.

On the termination (after the loop) we have that \( R \in SR \). Therefore, all the sets in \( R \) are already visited, and moreover, can be visited again by the same sequence of steps. The cycle is detected, from which the states in \textit{final} are reachable, and the returned value therefore is \( \infty \). \( \square \)

### 7.4.2 Computing Extremal Durations of QDDC Formulae

In the previous section we presented the algorithms for computing minimal/maximal duration of the interval between two states satisfying certain start and final condition respectively. Now we will show how the problem of computing extramal time bounds for a given QDDC formula may be expressed in terms of those algorithms. The approach is described in [115].

We denote the functions used to compute extremal durations of a QDDC formula \( d \) as \( MINDUR(d, \Sigma_\Delta) \) and \( MAXDUR(d, \Sigma_\Delta) \).

**Definition 7.4 (Extremal Duration Functions)**

\[
MINDUR(d, \Sigma_\Delta) = \min \{ \text{len}_\omega | \omega \in \text{subexec}(\Sigma_\Delta), \omega \models d \}
\]

\[
MAXDUR(d, \Sigma_\Delta) = \max \{ \text{len}_\omega | \omega \in \text{subexec}(\Sigma_\Delta), \omega \models d \}
\]

When there is a sub-execution of unbounded length, the \( MAXDUR \) function returns \( \infty \). If there is no any sub-execution that satisfies the formula, the function returns 0. The \( MINDUR \) function, in turn, returns \( \infty \) when there is no execution satisfying the formula.

Let \( st \) be a fresh boolean variable and \textit{AnyOnce}(\( st \)) be a transition system which nondeterministically sets \( st \) to true for at most one position in each of its execution (Fig. 7.4). Given a QDDC formula \( d \), and a transition
7.4. QUANTITATIVE ANALYSIS

system $\Sigma_\Delta$, we consider a new transition system $\Sigma'_\Delta$ defined as:

$$\Sigma'_\Delta = \Sigma_\Delta \times \text{AnyOnce}(st) \times \text{A}(true \, \neg \neg [st]^0 \, \neg \neg d)$$  \hspace{1cm} (7.1)

Let $end$ be a fresh boolean variable that is true when the automaton $\text{A}(true \, \neg \neg [st]^0 \, \neg \neg d)$ is in its final state. The following theorem justifies the presented approach:

**Theorem 7.3** Consider a GTS $\Sigma_\Delta$ and a QDDC formula $d$. Let $\Sigma'_\Delta$ be defined as in 7.1. Then,

$$MINDUR(d, \Sigma_\Delta) = \min_{\text{interval}}(st, end, \Sigma'_\Delta)$$,

$$MAXDUR(d, \Sigma_\Delta) = \max_{\text{interval}}(st, end, \Sigma'_\Delta)$$.

**Proof.** It is easy to see that the behavior of $\Sigma'_\Delta$ describes all and only the executions $\omega$ of $\Sigma_\Delta$, on which the following holds:

- $\omega$ starts at some initial state of $\Sigma_\Delta$;
- $\omega \models true \, \neg \neg d$.

Indeed, the variable $st$ is fresh and therefore does not restrict the set of possible executions. Therefore, $MINDUR(\Sigma_\Delta, d) = MINDUR(\Sigma'_\Delta, d)$, and $MAXDUR(\Sigma_\Delta, d) = MAXDUR(\Sigma'_\Delta, d)$.

By definition of $\text{AnyOnce}(st)$, $st$ appears on any sequence at most once. Therefore, for any $\omega \in \text{subexec}(\Sigma'_\Delta)$, $\omega \models d$ if and only if $\omega, 0 \models st$. On the other hand, for each $\omega \in L(\text{A}(true \, \neg \neg [st]^0 \, \neg \neg d))$, $\omega, \#\omega - 1 \models end$. Therefore, if $\omega \models d$ then $\omega, \#\omega - 1 \models end$.

As a result, we have that the duration of interval, where $d$ holds, is equal to the interval from $st$ to $end$, which implies the hypothesis. □
7.5 Discussion

We presented an extension to the analysis framework that intends to deal with time aspects in Web service compositions. For these purposes we presented notations for modelling time-related properties different from those covered by the standard Web service specification languages. These properties enable modelling of activity durations in business processes, constraining timing characterization of execution intervals, etc. Based on this model, we proposed an analysis approach that allows for verification of non-timed properties under timed constraints, validation of timed properties, and finally computation of time bounds, in which the properties hold. A discrete-time representation of the formal model is given in order to carry out the analysis tasks automatically.

As yet, only few works address the time-related issues in Web service compositions. Indeed, the complexity of the analysis drastically increases in the presence of time. However, these issues are particularly important for the long-running processes, such as e-government and supply-chain management applications, since they affect the correctness and even control flow of the composition behavior.

The work of [60] represents a timed formalisation of BPEL compositions, and defines a formal mapping to a network of timed automata. However, [60] does not allow for the explicit high-level definition of time-related assumptions and requirements. The properties are defined in less expressive form and should be specified directly in the UPPAAL model checker specification, complicating the composition description and analysis.

In [11] temporal abstractions are exploited for the compatibility and replaceability analysis of Web service protocol. In that model one can specify when certain transitions must or may happen, similarly to what we achieve with our duration annotations. However, [11] does not address
the problem of the verification of these time properties. Moreover, their temporal abstraction are much simpler with respect to the set of properties we can express in our approach.

The work close to that is the one presented in [41]. The authors present a framework for the analysis of timed properties in WS-CDL specifications. The analysis is performed by translating the specification to a set of timed automata. The authors present a goal-oriented notation for representing time requirements in terms of CTL queries. However, the restrictions remain: the properties require an explicit reference to clock variables in order to describe quantitative properties. Opposite to [41], our formalism hides these details from the designer, thus improving the quality and usability of modelling. Also, in [41] constraining of the specification with timed properties is not allowed.

The duration calculus logic (more precisely, its subset) presented here is a very expressive formalism for modelling and model checking timed properties [28]. In general, the logic is undecidable, but a lot of expressive yet decidable subsets were defined. Our discrete-time formalisation is a subset of the QDDC logic, for which an automata-theoretic decision procedure exists. This formalisation allowed us still to use NuSMV model checker and to easily define the quantitative algorithms for computation of extremal bounds for (QD)DC formula. The problem of discrete-time version is that in the worst case the size of the automaton for the formula is non-elementary. This, however, rarely happens in practice [113].

There are also decidable dense-time subsets of DC. Among them LIDL [114] is a notation, to which our model may be reduced under reasonable restrictions. A formula of LIDL may be translated into a timed automaton, which would inspire the use of corresponding model checkers, such as UPPAAL. A detailed investigation on the dense-time model, as well as performance comparison of the approaches is left as a future work.
Chapter 8

Implementation and Evaluation

The analysis techniques and approaches presented in previous chapters were implemented as a prototype toolkit, namely WS-VERIFY. The toolkit was realized within and is available as part of the project Astro [7]. The tool provides the analysis and the translation capabilities that instantiate the described approaches and generate the corresponding model checker specifications. In this chapter we describe the tool and its implementation in details, providing an insight into the supporting optimization techniques. These techniques turned out to be vital for the efficient verification of real-world case studies and applications.

In order to evaluate the analysis framework, as well as the presented techniques and algorithms, we conducted a range of experiments applied to the case studies described in the thesis. In this evaluation we exploited the WS-VERIFY toolkit in order to automatically translate the composition models to the model checker specifications, and to perform certain kinds of analysis internally. Here we discuss the results of the experiments conducted to justify the approach to the analysis of communication models, the presented data flow analysis approach, and finally the verification and computation of timed properties in Web service compositions.
8.1 WS-VERIFY Tool

The WS-VERIFY tool presented here is implemented as a Java API that extends the translator facilities of the ASTRO toolkit with the capability to preprocess, analyze, and emit the specification acceptable by the model checker. The presented analysis techniques, and the efficiency of the verification tasks require the extensive use of various optimization algorithms. Here we present the tool architecture, and describe the optimizations used in the tool implementation.

8.1.1 Architecture

The functional architecture of the WS-VERIFY is presented in Fig. 8.1. The execution of the tool passes several steps, where the composition model is sequentially transformed.

Input specification

In its current version, the tool accepts the composition model of the following form.

- The composition descriptor file defines the participants, providing references to their local specifications. The local specification is presented as a BPEL file that describes the behavior of the participant, and a set of WSDL files that describe the interface of the participant and of its partners.

- This default specification is enriched with a set of various properties, including BPEL duration annotations and complex time requirements; data flow properties that constraint the functions used in the specifications; behavioral requirements to be verified, that is, possibilities and assertions.
Additionally, the communication model should be specified. It is possible to use the adequacy analysis to automatically determine an appropriate model, or to use a predefined one, if some a priori knowledge about the interactions is provided.

Analysis parameters that allow to enable/disable a certain analysis technique, to select a certain data model (e.g., bounded data domains or a model of abstraction), etc.

This set of inputs may be extended in order to accept other composition formats, such as WS-CDL, and types of analysis, e.g., conformance checking.

**Tool execution**

The tool functionality is performed in several steps. Depending on the specified parameters, certain steps may be omitted.

1. The set of local models is translated into the set of state transition systems. The functionality is provided by the special module of the ASTRO toolkit, namely WS-TRANSLATOR. Syntactical errors, data types, and expressions are checked at this step.
8.1. WS-VERIFY TOOL

a) If specified, the adequacy analysis is applied in order to determine an appropriate communication model (Fig. 8.1.a). This analysis can detect potential buffer overflows, where the queue content grows unboundedly, and the composition incompleteness, i.e., message losses. In this case, the descriptions of the “bad” executions are reported to the designer together with the adequate communication model, and the used queue bounds. The latter is also exploited for further analysis/translation.

2. Given a set of STSs and a communication model, the corresponding state transition system with queues is constructed. The global graph of CSTS is built using DFS traversal algorithm. We remark, that in order to avoid state space explosion problem and perform the construction efficiently, we apply several optimization techniques here, namely partial order, and cone-of-influence reductions. This entails also the necessity to generate a separate model for each verification property, since the latter is used in the partial order reduction algorithm.

b) In the next step the data flow is managed. If the abstraction techniques are not used (i.e., data flow is ignored, or finite data ranges are applied), the reduced model is propagated to the later steps. Note, that the data constraints and functions are ignored in this case. If the abstraction-based analysis is applied, then the model is preprocessed and transformed according to the rules presented in Chapter 6. In particular, a minimal set of propositional variables is extracted; the transition conditions, properties, and assignments are modified accordingly.

3. At this phase, the control and data parts of the model are made finite. If the timed analysis is skipped, then the final model checker specification may be created. Otherwise, the following activity digitize and
translate the timed properties.

c) The timed properties of the composition are handled as follows. A set of (discrete) clock variables is extracted from the specification, as well as their upper bounds. The timed activities and activities annotated with duration constraints are transformed according to the rules described in Chapter 3 and 7. The complex requirements and constraints (in QDDC format) are translated into the finite state automata using the tool DCVALID [113].

4. The final composition models, together with the verified requirements and constraints, are emitted as model checker specifications. Currently, the NuSMV and SPIN formats are supported. However, while the NuSMV version allows for the full set of analysis/translation activities, the SPIN version is more restricted. In particular, abstraction-based analysis techniques and timed analysis are not supported. The investigations on the applicability of these techniques, potential encoding, and performance considerations are left for the future work.

Formal analysis

Using the model checker specifications generated by the tool it is possible to carry out verification activities. These activities currently include verification of behavioral and timed requirements expressed in LTL or DC respectively, and computation of extremal bounds for DC properties. The properties may be defined by the designer, or by using certain predefined patterns. The latter include (but, indeed, not limited to) deadlock and livelocks freeness verification, completeness of the composition (i.e., queue emptiness) if complained by the adequacy analysis, etc.

Currently, the tool does not provide any support for the simulation-based analysis, such as service replaceability or conformance checking.
While a variety of tools and algorithms for (both symbolic and explicit) simulation checking are presented in the literature, their applicability to the analysis of Web service compositions in presence of timed properties and asynchronous communications requires further investigation, and is left for future work.

8.1.2 Techniques and Optimizations

The formal model of the Web service compositions characterize the collaborative behavior of concurrently executed processes that interact with each other exchanging data. High degree of concurrency, the necessity to record and operate a non-trivial data structures, and the timed nature of wide range of typical applications considerably complicate the representation and the analysis of the systems leading to a state space explosion problem in the verification of the real-world applications. In order to tackle this problem, we utilize a set of techniques that aim at reduction and efficient exploration of the state space, without affecting the analysis results.

Control graph reduction

The Web service compositions are essentially asynchronous systems, i.e., the activities taken by the participating services are executed independently. In the global view of the composition this results in the interleaving model, where concurrent activities appear in arbitrary order. This leads to an extremely large number of states/transitions when all possible interleavings are considered.

The partial order reduction [118] allows to decrease the number of interleaving sequences that should be considered. This technique groups sequences in the set of all the interleavings into the equivalence classes: the sequences of states that have the same effect for a given property are
Table 8.1: Partial order reduction

<table>
<thead>
<tr>
<th>Case study</th>
<th>Non-reduced</th>
<th>reduced</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>States</td>
<td>Transitions</td>
</tr>
<tr>
<td>VTA (4 participants)</td>
<td>630</td>
<td>1361</td>
</tr>
<tr>
<td>VTA (7 participants)</td>
<td>34285</td>
<td>131089</td>
</tr>
<tr>
<td>VTA (cancellation)</td>
<td>1133</td>
<td>2361</td>
</tr>
<tr>
<td>Extends Loan Approval</td>
<td>750</td>
<td>1967</td>
</tr>
<tr>
<td>WMO</td>
<td>47365</td>
<td>206532</td>
</tr>
</tbody>
</table>

grouped together. Only one representative for a given class needs to be checked and stored, thus greatly reducing the state space of the model.

The technique is applied as follows. The global transition system is constructed by the depth-first search algorithm. At each step, only a subset of all the transitions leaving a current global state is considered. This subset, referred to as an ample set, is determined based on the transition independence and visibility with respect to the verified property [35]. The absence of the shared variables and predefined queue structure allow to compute the ample set efficiently, with a little overhead in the worst case. The reduced composition is constructed on-the-fly from the process skeletons. The data- and time-related properties of the states and transitions are exploited in order to compute ample sets correctly. The results in Table 8.1 show the importance of this technique for the analysis of service compositions.

The language we use to specify behavioral requirements, namely Linear-time Temporal Logic (without “next” operator), is compatible with the applied partial order reduction algorithm. This, however, does not necessarily hold for an arbitrary DC formula used to express timed properties of the composition. As in the case of LTL with the “next” operator, a DC formula may be affected by stuttering in the path. There are essentially two sources of such a problem in the DC syntax (more precisely, in the sub-
set used in our work): the measurement operators and the chop operator that relates two point operators. These constructs allow to count number of states in an interval, which is not compatible with the stuttering. We solve the first problem by making visible the timed transitions. The second problem may be solved by constraining the shape of the allowed DC expressions, or by finding a procedure to determine stutter-invariance. A deeper investigation on this issue is left for future work.

**Data optimization**

Another source of state space explosion is the data flow part of the model. Indeed, the state size increases with the growth of the number of variables and data domains. The challenging problem is therefore to find and extract a minimal representation of data and data manipulations.

The cone-of-influence reduction is a technique that allows one to extract a minimal set of variables that are relevant for the analysis of a given property. Intuitively, the reduction is performed as follows. Initially, the set of variables $vars_0$ consists of the variables mentioned in the verified property. Then it is extended with the variables participating in the transition conditions in order to ensure the correctness of the control flow. Finally, this set is iteratively augmented with the variables that affect those in $vars_i$. More precisely, given an expression $e$, let $vars(e)$ be the set of variables mentioned in $e$. Then, for every variable $v \in vars_i$ and for any assignment $v := e$ we require $vars_{i+1} = vars_i \cup vars(e)$. The iteration continues until a fix point is reached. In spite of its simplicity, this reduction may lead to a considerable improvement in the verification. In particular, for one of the examples the model checker was able to verify a property within 20 seconds, while without this optimization the result was not obtained even within an hour.

A similar optimization is applied in order to reduce the number of queue
variables. The idea is to check whether the variable used to instantiate the message is modified before this message is consumed. If there is no such modification, the intermediate queue variable is redundant and may be omitted in the model. Otherwise, such a variable is necessary in order to keep track of the actual value passing.

The use of data abstraction techniques is essential for reasoning on infinite data domains, custom functions, and partial specifications. These techniques allow to finitely represent such systems, using a set of propositional variables and axioms that constrain their evolution. This, however, puts additional restrictions on the implementation of the verification engine. Indeed, such an approach implies a logic-based representation of the data flow, where the variables are replaced with facts, and the data operations are replaced with propositional formula that express their semantics. Symbolic model checking techniques and the corresponding model checkers allow to address these issues in the natural way. Symbolic algorithms operate Binary Decision Diagrams, canonical representations of propositional formula, and hence represent the analyzed system in the form of logics. On the contrary, the use of abstractions together with explicit model checkers, such as SPIN, is problematic. A partial representation of the data flow, axioms over data, and non-determinism is neither simple nor efficient.

8.2 Evaluation and Experimental Results

We evaluated the analysis approaches on a set of case studies from different domains. We used various implementations of the VTA case study presented in Chapter 5. These implementations differ in representation of the control flow (e.g., cancellation mechanisms, number of participating services), the data flow (the usage of internal functions), and the time flow (timeouts in cancellation). These variations were exploited to evaluate all
the techniques presented in the thesis, that is adequacy analysis and the verification under different communication models, the data and time flow analysis. The Extended Loan Approval composition (Chapter 6) and the Waste Management Office application (Chapter 7) were used solely for the evaluation of the data and time flow analysis respectively. We remark that although the case studies are relatively simple, they are still considerably more complex with respect to the examples presented in other tools (e.g., [56, 53]).

8.2.1 Analysis of Communication Models

We conducted series of experiments in order to evaluate the communication systems analysis approach presented in Chapter 5. We experimented with different scenarios in the VTA case study.

Determining an adequate communication model

We applied the adequacy analysis algorithm to determine an appropriate communication models and to verify some properties (e.g., termination in a correct state). The analysis results are summarized in Table 8.2. The table reports the size of the generated control flow graph of the problem, the results of the adequacy/completeness analysis, and the performance of the property verification in Nusmv model checker. The results of the adequacy analysis define the total time of the skeleton analysis (imple-
mented in Java) and the verification of the composition completeness in the presence of data in NuSMV. For the verification, the total state space is presented.

**Comparison of different communication models**

We set up a bunch of experiments to demonstrate that the less general model shown to be adequate is more efficient for the analysis, and to see the overall performance of the composition verification based on the presented approach. In particular, we were interested in the performance and in the memory usage of the composition analysis. In these experiments we used several variations of the VTA example, where the number of the participating processes grows from two up to seven processes. We remark that the VTA process also grows since it interacts with increasing number of services. The ranges of the domain types used in the messages (e.g. Flight, Time) were bound and set to three values for each type.

In order to compare the verification complexity on the same scenarios under different communication models, we have used domains where the synchronous model is adequate. We used two behavioral properties in the experiments. The first property (P1) requires that the user process terminates successfully (reaches state “SUCC”) only if also the reservation services do. It can be expressed as the following expression:

\[(\Diamond \text{INSTATE}(user, succ) \rightarrow \bigwedge_i \Diamond \text{INSTATE}(service_i, succ))\]

This assertion property is expected to be valid in the domain, i.e., to be respected by all the executions of the Web service composition.

The second property (P2) expresses the possibility for the partners to terminate successfully:

\[(\Diamond \text{INSTATE}(user, succ) \land \bigwedge_i \Diamond \text{INSTATE}(service_i, succ))\]
8.2. EVALUATION AND EXPERIMENTAL RESULTS

Table 8.3: Scalability: NuSMV results

<table>
<thead>
<tr>
<th></th>
<th>Sync</th>
<th>LO</th>
<th>UO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Size</td>
<td>Time</td>
</tr>
<tr>
<td>P1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.08</td>
<td>5</td>
<td>0.09</td>
</tr>
<tr>
<td>3</td>
<td>0.11</td>
<td>8</td>
<td>0.13</td>
</tr>
<tr>
<td>4</td>
<td>0.19</td>
<td>11</td>
<td>0.22</td>
</tr>
<tr>
<td>5</td>
<td>0.26</td>
<td>14</td>
<td>0.32</td>
</tr>
<tr>
<td>6</td>
<td>0.32</td>
<td>17</td>
<td>0.42</td>
</tr>
<tr>
<td>7</td>
<td>0.44</td>
<td>20</td>
<td>0.64</td>
</tr>
<tr>
<td>8</td>
<td>1.95</td>
<td>25</td>
<td>3.52</td>
</tr>
<tr>
<td>P2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.01</td>
<td>5</td>
<td>0.02</td>
</tr>
<tr>
<td>3</td>
<td>0.33</td>
<td>8</td>
<td>0.48</td>
</tr>
<tr>
<td>4</td>
<td>1.14</td>
<td>11</td>
<td>2.28</td>
</tr>
<tr>
<td>5</td>
<td>20.1</td>
<td>14</td>
<td>36.8</td>
</tr>
<tr>
<td>6</td>
<td>114</td>
<td>17</td>
<td>221</td>
</tr>
<tr>
<td>7</td>
<td>771</td>
<td>20</td>
<td>1279</td>
</tr>
<tr>
<td>8</td>
<td>&gt;1 hour</td>
<td>&gt;1 hour</td>
<td>&gt;1 hour</td>
</tr>
</tbody>
</table>

This property is expected to be satisfiable, i.e., there are some executions of the composition where the property is true. Moreover, we expect that the verification task produces a trace corresponding to one these executions, thus witnessing the validity of this possibility.

The results of the verification of these properties are summarized in Table 8.3 for NuSMV model checker [30] and in Table 8.4 for SPIN [71]. We tested the specifications of the composition under synchronous product (Sync), under locally order model (LO) and under the most general model (UO). The table contains information on the time used for the verification and counterexample generation in seconds, and on the size of the state vector in bytes. That is, if the state vector size is 14 bytes, the state space is $2^{8 \times 14}$ states.

Some comments on the difference in the performance of NuSMV between the two properties. This is due to the fact that the second property
requires the generation of a witness scenario, and this takes a lot of time. The time required by NuSMV to report that the second property can be satisfied without extracting the witness trace is similar to the time required to verify the first property. On the contrary, the verification using SPIN model checker requires much more time for the first property. Indeed, in this case all the behaviors have to be explored to prove the correctness of the property.

The presented results demonstrate the reduction of the verification performance when more general communication model is applied. This is explained by the fact that more general model introduces more queue variables and therefore increases the state space size. This factor is important for both model checkers (and similar to those): the analysis techniques strictly depend on the number of variables, their domains and relations. As a result, when the data domains grow (or the number of abstract vari-

---

Table 8.4: Scalability: SPIN results

<table>
<thead>
<tr>
<th></th>
<th>Sync</th>
<th></th>
<th>Sync</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Size</td>
<td>Time</td>
<td>Size</td>
<td>Time</td>
<td>Size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td>2</td>
<td>0.87</td>
<td>48</td>
<td>0.89</td>
<td>56</td>
<td>0.89</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.05</td>
<td>60</td>
<td>1.09</td>
<td>76</td>
<td>1.13</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.51</td>
<td>72</td>
<td>1.60</td>
<td>100</td>
<td>1.75</td>
<td>172</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3.67</td>
<td>84</td>
<td>4.79</td>
<td>120</td>
<td>6.37</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>90</td>
<td>100</td>
<td>&gt;1Gb</td>
<td>&gt;1Gb</td>
<td>&gt;1Gb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>0.87</td>
<td>48</td>
<td>0.89</td>
<td>56</td>
<td>0.92</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.05</td>
<td>60</td>
<td>1.09</td>
<td>76</td>
<td>1.13</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.38</td>
<td>72</td>
<td>1.46</td>
<td>100</td>
<td>1.52</td>
<td>172</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.85</td>
<td>84</td>
<td>1.85</td>
<td>120</td>
<td>1.95</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>2.31</td>
<td>100</td>
<td>2.33</td>
<td>140</td>
<td>2.46</td>
<td>272</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>2.89</td>
<td>112</td>
<td>2.98</td>
<td>160</td>
<td>3.27</td>
<td>320</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>3.99</td>
<td>124</td>
<td>4.36</td>
<td>184</td>
<td>5.23</td>
<td>372</td>
<td></td>
</tr>
</tbody>
</table>

requires the generation of a witness scenario, and this takes a lot of time. The time required by NuSMV to report that the second property can be satisfied without extracting the witness trace is similar to the time required to verify the first property. On the contrary, the verification using SPIN model checker requires much more time for the first property. Indeed, in this case all the behaviors have to be explored to prove the correctness of the property.

The presented results demonstrate the reduction of the verification performance when more general communication model is applied. This is explained by the fact that more general model introduces more queue variables and therefore increases the state space size. This factor is important for both model checkers (and similar to those): the analysis techniques strictly depend on the number of variables, their domains and relations. As a result, when the data domains grow (or the number of abstract vari-
ables used in abstraction), the difference between the performance results grows as well, by the factor exponential to the former.

### 8.2.2 Abstraction-based Data Flow Verification

The abstraction-based analysis of data-related properties was evaluated on the VTA and on the Extended Loan Approval case studies. In the VTA example we demonstrate a simple problem specific to the data flow analysis and evaluate the efficiency of the approach. A sophisticated analysis scenario, where the model of the composition is incrementally analyzed and refined, is presented for the Loan Approval example.

#### Virtual Travel Agency

In spite of its simplicity, the VTA case study may exhibit a serious problem that is related to data management and is not revealed by existing verification tools. In the example the application starts from the user request, where the desired time period and the location are specified as parameters of the corresponding (abstract) data types. The request is forwarded to the Flight and to the Hotel services. In the offer the former specifies the flight description and the time, while the latter specifies the hotel.

Now let us consider two different Flight services. The first is internally implemented in such a way that it provides an offer only if there exists a flight completely matching the requested time/location. The second one, instead, provides a closest offer if there is no exact match. In the context of Virtual Travel Agency the second service is not acceptable, since the incorrect offer makes the composite offer inconsistent. Note, however, that the information on the internal implementation may not be available, hence resulting in incorrect composition when the wrong service is used.

The data analysis approach presented in Chapter 6 addresses the problem as follows. In order to check the requirement that the offer matches
the request, the following assertion is verified:

\[ \square (\text{ACTION(offer)} \rightarrow \text{offer.time} = \text{request.time}) \]

When the property is verified on the over-approximated model, the violation is reported. The counterexample demonstrates a scenario, where the flight service provides an offer with the time different from that in the request, and therefore the offer returned to the user is incorrect as well.

The next step is to re-verify the property on the under-approximated model. The model checker does not reproduce the counterexample, which means the lack of information about the implementation. Indeed, the property is satisfied if the first Flight service is used, while violated by another one. This lack of information may be resolved by adding the following assumption on the Flight service internals:

\[ \text{flightTime(chooseFlight(fReq.place, fReq.time))} = \text{fReq.time} \]

That is, the time of the offered flight should be equal to the requested. Obviously, under this constraint the property is not violated. This constraint is then should be exploited when the appropriate Flight service is being chosen, or at run-time to monitor that the Flight service respects its contract.

Based on the VTA example, we also evaluated the performance of the data analysis approach. In particular, we compared the state space/time values of the abstraction-based analysis\(^1\) with those of the finite domain analysis (2, 4, and 6 values per type). The results are represented in Fig. 8.2. The left part demonstrates dependency between the state space and the number of participating services. The right parts shows an analogous dependency for the total analysis time (i.e., translation and verification). The

\(^1\)We remark that in the experiments only the results for the B-model are presented. Due to a sophisticated translation of the K-model, it is used only for the validation of the counterexamples generated by the B-model. However, the verification time/state space under K-model are similar to those of B-model.
size of the control graph ranges from 176/180 to 394/401 states/transitions. A total number of typed variables ranges from 10 to 63.

From the comparison follows that the abstraction-based analysis is as efficient as the verification under very small data domains. Moreover, the performance of the latter approach decreases drastically with the growth of the data range. Note also that, even if the analysis for the domains with two values is more efficient, the verification results may be incorrect. Indeed, if the variables have only two values, namely “defined” and “undefined”, the verification of the above assertion will result in a false positive answer.

We remark, that the above results may be improved if the cone-of-influence reduction is applied. However, the results obtained with the abstraction-based approaches remain more complete.

Extended Loan Approval

The Extended Loan Approval case study presented in Chapter 6 addresses more complex analysis scenario. There are several factors related to data that affect the composition behavior.

First, it depends on the range of possible amounts requested by the user.
(i.e., the values generated by the function \textit{initial}). Indeed, if only very small loans are allowed, then it is very likely that the Approver service will not be involved.

Second, it depends on the way the Approver and the Assessor services are implemented (i.e., functions \textit{approve} and \textit{assess}). These details are not necessarily known \textit{a priori}, and may differ from one implementation to another.

Third, the correctness of the behavior depends on the correctness of the data flow operations (message passing and assignments). The variables are extensively used in the example to guide the control flow (i.e., in the transition conditions).

During the analysis of the composition various properties are verified, and the model of the application is incrementally refined. We use both over- and under-approximated abstractions in order to verify properties and extract the required assumptions on the implementations. The results of the analysis are represented in Table 8.5. The following analysis steps are performed:

1. Verify a possibility to successfully complete the process. This is a kind of consistency check, since we verify the existence of an execution that demonstrates a nominal use case:

\[
\Diamond \text{INSTATE}(\text{customer}, \text{succ})
\]

While a witness for the possibility is found in the B-model, it is not replicated in the K-model of the composition. Indeed, since the valuation of functions \textit{initial}, \textit{approve}, and \textit{assess} are not known, the transition conditions that rely on these values can not be precisely evaluated and the execution is blocked.

2. In order to refine the composition model, we add data assumptions on the implementation of the functions. First, we allow the user to
8.2. EVALUATION AND EXPERIMENTAL RESULTS

request any loan amount (function \texttt{initial})\(^2\):

\[
true \rightarrow large(result) \sqcup small(result) \sqcup \neg large(result) \wedge \neg small(result)
\]

Second, we assume that the Assessor service assigns a low risk when the amount is small (function \texttt{assess}):

\[
(small(p)) \rightarrow (result = "low")
\]

Under these assumptions the property is satisfied in both models.

3. The witness trace obtained on the previous step demonstrates a case where the requested amount is small and the Assessor service assigns a low risk. Now we check a possibility to obtain an approval when the Approver service is involved:

\[
(\Diamond approver.resp.answer = "yes") \wedge (\Diamond \text{INSTATE}(customer, succ))
\]

Again this possibility is satisfied in the B-model, but not in the K-model: the behavior of Assessor for large amounts is unknown, as well as the behavior of the Approver service.

4. The model is refined in order to satisfy previous property. We extend a constraint on the Assessor implementation such that it assigns a low risk if and only if the requested amount is small (function \texttt{assess}):

\[
(small(p)) \leftrightarrow (result = "low")
\]

Then, we assume that the Approver rejects only the large amounts (function \texttt{approve}):

\[
(large(p)) \leftrightarrow (result = "no")
\]

Under these assumptions the possibility indeed holds.

\(^2\)Here predicates \texttt{large(p)} and \texttt{small(p)} are used to represent, for instance, the constraints \(p \geq 10000\) and \(p \leq 100\) respectively.
5. Another important property to verify is the assertion that the application eventually terminates and the user is provided with an answer:

\[ \Diamond \text{INSTATE}(user, succ) \]

The property is violated under the B-model: there exists an infinite loop, where the Approver service rejects the request, but the amount remains sufficiently large. Indeed, in real execution the value should be eventually reduced, thus leading to an approval. Consequently, this violation is not replicated in the K-model.

6. The above problem is removed as follows. The composition model is equipped with an assumption on the function \textbf{reduce}, e.g.:

\[
\begin{align*}
\text{large}(p) &\rightarrow \neg \text{large}(result) \\
\neg \text{large}(p) &\rightarrow \text{small}(result)
\end{align*}
\]

Under these simple assumptions the assertion holds for both models.

7. The last property we check here is the assertion on the value returned to the customer. More precisely, this value may be less than the requested one if and only if the Approver server is called and if it rejects the request at least once:

\[
\Diamond(\text{ACTION}(\overline{response}) \wedge response.amount \neq request.amount) \leftrightarrow \Diamond(\text{ACTION}(\overline{answer}) \wedge answer = \text{"no"})
\]

This assertion is failed in both models. There exists a scenario, where the initial amount is first rejected, then reduced, and finally accepted by the Approver service. The user instead receives a value of the variable \textit{result}, instead of \textit{amount}, which is actually approved. As a result, the reduction has happened, but the returned value is the same as requested and the property is violated.
8.2. EVALUATION AND EXPERIMENTAL RESULTS

<table>
<thead>
<tr>
<th>Step</th>
<th>Abstraction</th>
<th>Result</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B-model</td>
<td>true</td>
<td>0.19 sec</td>
<td>6.21 × 10^{12}</td>
</tr>
<tr>
<td></td>
<td>K-model</td>
<td>false</td>
<td>0.16 sec</td>
<td>3.73 × 10^{11}</td>
</tr>
<tr>
<td>2</td>
<td>B-model</td>
<td>true</td>
<td>0.37 sec</td>
<td>5.60 × 10^{16}</td>
</tr>
<tr>
<td></td>
<td>K-model</td>
<td>true</td>
<td>0.28 sec</td>
<td>2.25 × 10^{17}</td>
</tr>
<tr>
<td>3</td>
<td>B-model</td>
<td>true</td>
<td>0.73 sec</td>
<td>1.13 × 10^{17}</td>
</tr>
<tr>
<td></td>
<td>K-model</td>
<td>false</td>
<td>0.16 sec</td>
<td>1.12 × 10^{12}</td>
</tr>
<tr>
<td>4</td>
<td>B-model</td>
<td>true</td>
<td>2.29 sec</td>
<td>3.59 × 10^{18}</td>
</tr>
<tr>
<td></td>
<td>K-model</td>
<td>true</td>
<td>0.41 sec</td>
<td>4.93 × 10^{20}</td>
</tr>
<tr>
<td>5</td>
<td>B-model</td>
<td>false</td>
<td>1.47 sec</td>
<td>1.43 × 10^{19}</td>
</tr>
<tr>
<td></td>
<td>K-model</td>
<td>true</td>
<td>0.46 sec</td>
<td>4.93 × 10^{20}</td>
</tr>
<tr>
<td>6</td>
<td>B-model</td>
<td>true</td>
<td>2.88 sec</td>
<td>2.29 × 10^{20}</td>
</tr>
<tr>
<td></td>
<td>K-model</td>
<td>true</td>
<td>0.66 sec</td>
<td>1.34 × 10^{22}</td>
</tr>
<tr>
<td>7</td>
<td>B-model</td>
<td>false</td>
<td>7.30 sec</td>
<td>1.18 × 10^{23}</td>
</tr>
<tr>
<td></td>
<td>K-model</td>
<td>false</td>
<td>0.74 sec</td>
<td>3.23 × 10^{24}</td>
</tr>
<tr>
<td>8</td>
<td>B-model</td>
<td>true</td>
<td>15.32 sec</td>
<td>1.19 × 10^{23}</td>
</tr>
<tr>
<td></td>
<td>K-model</td>
<td>true</td>
<td>1.08 sec</td>
<td>3.28 × 10^{24}</td>
</tr>
</tbody>
</table>

8. The above problem should be resolved by changing the specification of the Loan Approval service: the variable sent to the user should be assigned the actual value (as it is done when the Assessor service provides a low-risk result).

8.2.3 Validation and Computation of Timed Properties

We illustrate the timed analysis using two case studies, the VTA with cancellation and WMO. The first case study demonstrates typical transactional scenario: the user should have a maximal freedom to cancel the order, while minimal reservation delays of the partners, together with the state consistency are guaranteed. In the WMO example, however, the main focus is on the ability to negotiate different timing constraints in a process with a simple control flow but a complex timed structure. For these case
studies we consider both the qualitative and quantitative analysis, and evaluate the performance.

VTA with cancellation

The VTA case study was used for the experiments on the timed analysis demonstrates the following scenario (Fig. 8.3). The user sends a request for hotel/flight reservation. The VTA service invokes both Hotel and Flight reservation services in order to obtain the corresponding offer. Having provided an offer the Flight service waits for a certain time for confirmation/cancellation. If this time expires (activity \texttt{onAlarm}), the reservation is only cancelled. The Hotel and VTA services are implemented analogously.
The behavior of this composition depends on the timed parameters of the specification. Indeed, the confirmation of the user may result in a successful ordering, in a cancellation of the transaction, or even in a failure of the process. The latter critical run occurs when the user confirms the offer, the VTA and flight accept the confirmation, but the hotel service rejects it due to timeout. It is easy to see that the occurrence of these scenarios is strictly related to the response time of the user (i.e., duration of the decision activity) and to the values of the timeouts of the services (i.e., parameters $F$, $H$, and $V$).

We verified the following properties under different settings:

- There is no execution leading VTA to a failure (Assertion $P1$):
  \[ \Box \neg \text{instate}(vta, \text{failure}) \]

- If User confirms, a ticket is eventually received (Assertion $P2$):
  \[ \Diamond \text{action}(\text{confirm}) \rightarrow \Diamond \text{action}(\text{ticket}) \]

- $P3$: It is possible to buy a ticket (Possibility $P3$):
  \[ \Diamond \text{action}(\text{ticket}) \]

The verification results are presented in Table 8.6. The size of the skeleton control graph is 335-390 states, 405-460 transitions. The decision
time of the user is not constrained. The parameters $H$, $F$, and $V$ represent the values of timeouts for the Hotel, Flight, and VTA services respectively. Here one can see that if the timeout of the Hotel is less than timeout of the Flight and VTA services, the failure state is reachable, otherwise it is unreachable.

Another set of experiments was devoted to the quantitative analysis. In particular, we computed the minimal/maximal duration of the user decision, such that the following properties guaranteed: the confirmation always results in a ticket emission (property C1); the confirmation always results in a cancellation (C2): the confirmation always results in failure (C3). The analysis is performed as follows. First, the property (C1–C3) is added to a specification as a constraint. Second, the min/max algorithms are applied to the constrained model to measure the following property:

$$\lceil \text{ACTION(ofer)} \rceil^0 \triangleright \text{true} \triangleright (\lceil \text{ACTION(confirm)} \rceil^0 \lor \lceil \text{ACTION(cancel)} \rceil^0).$$

The property describes an interval that corresponds to the user decision. The results are presented in Table 8.7. From the experiments follows, for instance, that the failure state is reachable, if the User confirms the order after the timeout of the Hotel service, but before the timeout of the Flight service. Indeed, this is not possible if the timeout of the Flight is smaller than those of the Hotel service.

From the experiments also follows that the analysis performance in our implementation (i.e., discrete-time model with integer clocks) depends
on the dimension of the time domain. This dependency is described in Fig. 8.4: the verification time (left) and the number of reachable states (right) are represented. Note that this dependency does not hold for the approaches, where specialized methods for the state space representation are used (see [143] for a complete survey). We expect analogous results in our approach, if similar techniques are applied. In particular, in [127] a method for efficient representation (and reasoning) of discrete timed automata in NuSMV is represented. Based on this method, the performance results presented here may be considerably improved, without affecting the quantitative algorithms. Deeper investigation on these techniques is left for future work.

Waste Management Office

We verified the behavior of the Waste Management application (Fig. 7.1) against various properties of interest, and under different assumptions expressed using both the duration annotations and DC expressions.

The WMO procedure includes several time consuming activities: “Eval-
uate Documents” (ED), “Provide Integration” (PI), “Technical Analysis” (TA), “Verify Reviews” (VR), “Change Date” (CD), “Conference” (CONF), “Provide Decision” (PD). One of the goals of the analysis is to determine proper durations for these activities such that the global requirements are satisfied, without affecting local requirements.

According to the official procedure, the process should satisfy the following global requirements:

- The notification of the Procedure Manager, the expert analysis, and the first conference call should happen no later than 4 weeks after the registration of request (R1):
  \[ \square([\text{reg}]^0 \rightarrow \true ( [\text{notif}]^0 \lor [\text{invokeTC}]^0 \lor [\text{confCall}]^0) \Rightarrow \text{len} \leq 4); \]

- The conference announcement should precede the conference itself at least 2 weeks (R2): \[ \square([\text{confCall}]^0 \rightarrow \true \Rightarrow \text{len} \geq 2); \]

- The interval between the conference announcement and the publication of conference documentation (acts) should not exceed 8 weeks (R3): \[ \square([\text{confCall}]^0 \rightarrow \true \Rightarrow \text{len} \leq 8); \]

- The whole procedure should not exceed 12 weeks (R4): \[ \text{len} \leq 12. \]

In order to ensure that the conference date may be changed only within 1 week after the announcement, the Procedure Manager process is equipped with a timeout event handler, whose activity is fired if the user does not provide a new date within this period.

Given these requirements, we incrementally analyzed the composition. The results are represented in Table 8.8. The following analysis steps were performed:

1. **Consistency checking.** The first analysis step consists of verification that the global requirements are consistent with each other and the
application process. This is checked as follows: the global requirements are added to the specification as constraints, and the behavior of the resulting system should contain a complete system execution. The time consuming activities are not constrained in this case (i.e., $\text{dur}(a) \geq 0$).

2. **Consistency of the global and local constraints.** The next step is to add local duration constraints to the specification and verify that the requirements are still consistent. Initially, these constraints are defined as follows: $\text{dur}(ED) \geq 2$, $\text{dur}(TA) \geq 2$, $\text{dur}(VR) \geq 2$, $\text{dur}(CONF) \geq 3$, $\text{dur}(PD) \geq 2$. In other words, the offices reserve a certain time for their operations. Under these constraints the specification is inconsistent. Therefore, some of the lower bounds should be decreased: $\text{dur}(ED) \geq 1$, $\text{dur}(TA) \geq 1$, $\text{dur}(VR) \geq 1$. The tool reports consistency in this case.

3. **Bounding time consuming activities.** While the previous step ensures consistency of the requirements and the lower time bounds of the activities, it is easy to see that the global requirements are violated when the upper bounds are not constrained. Indeed, if the $ED$ activity lasts for 5 weeks, the requirement $R_1$ is violated. For instance, the tool reported violation of the property $R_4$ for the following upper bounds: $\text{dur}(PI) \leq 1$, $\text{dur}(ED) \leq 2$, $\text{dur}(TA) \leq 3$, $\text{dur}(VR) \leq 2$, $\text{dur}(CONF) \leq 5$, $\text{dur}(PD) \leq 3$. Therefore, it is necessary to set certain upper bounds on the activities, such that all the requirements are satisfied. These bounds may be set manually or, in certain cases, may be extracted automatically, using min/max bounds algorithm. For the bounds $\text{dur}(ED) = 1$, $\text{dur}(TA) = 1$, $\text{dur}(VR) = 1$, $\text{dur}(CONF) = 3$, $\text{dur}(PD) = 2$ the property is satisfied.

4. **Computing quantitative parameters.** When the local constraints are
defined such that the global requirements are satisfied, the quantitative analysis may be applied to extract certain properties of the model. For instance, it is possible to compute the minimal and maximal duration of the whole procedure or its part defined using a DC expression. This information may be used to improve the bounds on the duration of activities obtained in previous step.

### 8.3 Discussion

In this chapter we reported practical results obtained during the application of the presented analysis framework to different case studies. The prototype implementation of the framework, the WS-VERIFY tool, incorporates different analysis capabilities, namely the analysis of communication models, abstraction-based data flow analysis, and the verification and computation of timed properties of the service compositions. These capabilities are provided by the tool itself (e.g., adequacy analysis and abstraction construction), and by the utilization of external tools, such as NuSMV and SPIN model checkers, and the DCVALID tool. The results of the experimental evaluation demonstrate the vitality of the presented
8.3. DISCUSSION

approaches. First, applying the presented techniques we were able to efficiently verify a range of non-trivial case studies. We remark, that the examples we used in the experiments are considerably more complex with respect to those found in other related works. This efficiency, however, was made possible by combining various reasoning and optimization techniques, and by the intensive engineering process. Moreover, the presented evaluation allowed us to verify much wider class of compositions and their properties. The ability to reason on various communication models, to represent and verify data and time aspects goes beyond capabilities of other existing tools, such as WSAT\(^3\) or WS-Engineer\(^4\).

Finally, we demonstrated how the presented analysis approaches may be applied to the design process of Web service composition in a methodological way. We instantiated the analysis of communication models on various versions of the VTA case study. We illustrated the analysis of data-related problems using the Extended Loan Approval example. The system was iteratively analyzed, refined, enriched with the assumptions on missing knowledge, and finally modified to correct the discovered problems. In a similar way we experimented with the approach to the analysis of timed properties. We applied this approach to the WMO application, where the timed requirements were modelled, negotiated, adjusted and computed using the techniques presented in this dissertation.

\(^3\)Web Service Analysis Tool. http://www.cs.ucsb.edu/~su/WSAT

\(^4\)http://www.ws-engineer.org
Chapter 9

Conclusions

The presented dissertation studies the problem of the formal representation and analysis of Web service compositions. We have developed an analysis framework that supports the composition design process with an ability to detect possible flaws in the specification before the composition is being actually deployed. This framework integrates formal specification and formal reasoning capabilities, defining a formal model for the representation of Web service compositions, and a set of formal methods and algorithms for the analysis of their behavior.

We build our framework on top of a formal model, where each participating service is given as a state transition system, used to represent the dynamic aspects of the service behavior. This formalism was developed with an aim to represent and describe (i) the control flow (i.e., evolution of a system that moves from a state to another performing some actions); (ii) the data flow (i.e., the business data of the services, its modification and exchange during interactions); and (iii) the time flow (i.e., the time properties, such as duration of actions, time-related events, such as time-outs and deadlines) of the composition. In this formalism the participants interact with each other through a formal model of the message delivery medium, namely communication model. The communication model de-
fines a set of (potentially unbounded) message queues, their structure and ordering constraints. The resulting composition is parametric with respect to these features allowing one to describe a wide range of asynchronous composition scenarios, middleware and protocol implementations.

We demonstrate how a standard specification of a Web service composition, such as a WSDL and BPEL models, may be represented in our formalism. We define a mapping of BPEL constructs, discuss the expressiveness and restrictions of our model in these regards. In order to model behavioral composition requirements, we exploit a linear-time temporal logics that allows for representation of a wide class of relevant properties and scenarios. Furthermore, we also provide a language for the representation of quantitative timed properties of the composition behavior. With this formalism we extend the qualitative analysis of the composite behavior, and provide a way to perform quantitative analysis, in particular computation of extremal time bounds for the given time properties.

The formal analysis approaches incorporated in the presented framework are based on the model checking techniques, and aim at providing a rigorous methodologies for incremental verification, validation, and refinement of the composition models. These approaches focus on a set of problems that are specific to the Web service compositions and play a very important role for their analysis:

- Analysis of asynchronous interactions. The message exchanges occurring in Web service compositions are essentially asynchronous and complex, the composite behavior may depend on the way the underlying queueing system is modelled and implemented. The use of parametric communication model allows us to uniformly represent and analyse the behavior of the composite system in different settings. We presented an approach that is able to identify the simplest communication model that puts the least constraints on the middleware im-
plementation, completely describes the behavior of the composition, and allows for the most efficient verification of the composition specification. The approach allows also to detect potential message losses and unbounded queue growth; the obtained results are then used to build a finite and efficient representation for further analysis tasks, i.e., model checking.

- Data flow analysis. We present an iterative analysis and refinement process that takes into account data flow of the composition. The approach is based on abstraction techniques, and allows to finitely represent and verify infinite data domains and complex data structures. It also provides a way to manage specifications, where the data management operations are represented only partially. This incompleteness of knowledge is treated through an iterative procedure, where the verification of requirements is interleaved with the elicitation of the assumptions on the missed knowledge that ensure these properties. As a result, the specification is enriched with a set of data-related properties that are crucial for the system correctness.

- Timed analysis. Facing the necessity to model and analyze long-running business processes and compositions, we extend the modelling and analysis capabilities of our framework, and provide an approach to deal with time-related characteristics of the composite behavior. A set of specific notations and algorithms is developed in order to perform both qualitative and quantitative analysis of timed properties and requirements. We remark that the presented approach allows for modelling these aspects at the level of business processes, hiding the translation procedure and low level details from the composition designer.

The analysis approaches presented in the dissertation are implemented
and incorporated into a prototype tool WS-VERIFY. The tool allows for automated mapping of BPEL compositions into the presented formalism. The composition model is augmented with additional properties, such as requirements specifications, data- and time-related properties. The resulting model is then analyzed with the presented techniques, and the verification of behavioral requirements is performed using state of the art model checking tools, namely NuSMV and SPIN. Using the prototype implementation, we evaluated our framework on a set of real-world case studies emerged from industrial applications and found in the literature. The results of the experiments demonstrated the viability and scalability of our analysis techniques, and allowed us to efficiently verify and identify some non-trivial problems in the case studies.

9.1 Directions for Future Work

There is a wide range of future research directions to be investigated. These directions would potentially address many important problems both from the methodological and from the technical points of view. We consider them as future extensions of the presented analysis framework and the WS-VERIFY toolkit.

In this dissertation we mainly concentrate on the bottom-up approach to the Web service composition design, and discuss the corresponding analysis techniques and methods. In particular, the composition is given as a set of independently specified service specifications, while the composition requirements are specified as a set of behavioral properties the composite model should satisfy. One of the possible extensions of the framework would be the development and implementation of the approaches that support the top-down analysis of the composition. In such an analysis one should verify the local implementations against the global specifica-
tion, perform the replaceability and composability analyses. In our works [82, 83] we have presented some preliminary results in this direction, where the presented formalisms are adapted in order to deal with global composition models. Apart from the implementation of these results, a lot of theoretical questions should be addressed. In particular, it is necessary to investigate the impact of the presence of complex data and time flow; to understand how the service replaceability analysis is affected by various asynchronous communications models.

Another set of extension would target the expressiveness of the presented formalism and the restrictions on mapping of standard Web service specifications, as those discussed in 4.1.2. In these settings, one of the most critical problems concerns the representation of dynamic (sub)process creation, message interaction correlations, dynamic service binding, etc. While these aspects are important for the composition design, their formalization and analysis is made problematic if at all possible. Indeed, such models lead to infiniteness of the composition reachability graph even if the data and time aspects are abstracted away. One of the possible solutions would be to restrict the (maximal) number of the created processes, leading to a priori finite representation. However, the results obtained in this way are essentially incomplete. Moreover, the state space explosion due to the increment of the process number drastically reduces the analysis performance. As a result, new methods and approaches, such those presented in [54], are needed in order to approach this problem.

We also work on the extension of the presented analysis techniques, i.e., on the analysis of communication models, verification of data-related and time-related aspects of the composition behavior. For what concerns the analysis of asynchronous communications, we plan to avoid using an explicit hierarchy of models, and construct a minimal adequate model on-the-fly. This would allow to improve the analysis performance, to obtain
more precise answer, and to determine particular message reorderings that are critical for the differences in behavior.

With respect to the analysis of the data flow, we are interested in the automated support for the analysis and extraction of the assumptions on incomplete composition specifications. In other words, when a precise answer for the verification of a certain property can not be obtained, the tool should be able to automatically provide the designer with certain assumptions that remove the lack of knowledge. Such an information would considerably simplify the iterative analysis of data-intensive compositions.

With respect to the timed analysis, there are several directions for future research. First, we are interested in application of dense-time semantics of the behavioral model, using e.g., LIDL [114] notation, to which our model may be reduced under reasonable restrictions. A formula of LIDL may be translated into a timed automaton, which would inspire the use of corresponding model checkers, such as UPPAAL. Second, we work on the performance of the timed analysis using specific approaches, such as state space clustering.

Last, but not least, we plan to work on improving the efficiency of the presented analysis techniques and approaches. The current results obtained with both the NuSMV and SPIN model checkers, may be improved using the special techniques, such as symbolic verification based on the Presburger arithmetic, native data abstraction techniques, etc.
Bibliography


[34] E. M. Clarke, J. M. Wing, R. Alur, R. Cleaveland, D. Dill, A. Emerson, S. Garland, S. German, J. Guttag, A. Hall, T. Henzinger,


