klase

Verification of Web Services

ITC-irst, Università di Trento, Università di Roma

Abstract. One of the key ideas behind the Web service technology is the ability to compose pre-existing services and systems into new distributed business processes and applications. The quality and the correctness of the composition requires continuous analysis of its behavior and validation of various requirements of the stakeholders, introducing new challenges that are peculiar to the Service-Oriented Architecture. In this work we present a formal analysis framework that allows for the verification of Web service composition facing these challenges in a rigorous and structured way.

Document Identifier	Deliverable D3.3
Project	MIUR-FIRB project RBNE0195K5 "Knowledge Level Automated Software Engineering"
Version	v1.0
Date	October 31, 2006
State	Final
Distribution	Public

Acknowledgements.

This document is part of a research project funded by the FIRB 2001 Programme of the “Ministero dell’Istruzione, dell’Università e della Ricerca” as project number RBNE0195K5.

The partners in this project are: Istituto Trentino di Cultura (Coordinator), Università degli Studi di Trento, Università degli Studi di Genova, Università degli Studi di Roma “La Sapienza”, DeltaDator S.p.A..

Executive Summary

The Service-Oriented Architecture based on the Web service technology emerged as a natural consequence of the evolution of distributed computing. One of the key idea of this technology is the ability to create service compositions by combining and interacting with pre-exisiting services using standard languages and protocols for the definition of the composition behavior. These composite business applications often span across the enterprise boundaries and involve various stakeholders into the design process, requiring negotiation of potentially inconsistent or even contradictory business requirements. The quality of the composed system and the success of its procurement, strictly depend on the ability to ensure that the composition addresses these requirements and resolves the conflicts. This makes the formal modelling and analysis techniques toolkit an essential part of the service-oriented design and development.

Providing an analysis support for the composition design, it is important to take into account the features peculiar to the Web services and SOA, which require specific approaches and techniques. Among them the problems of modelling and reasoning on service interactions and interaction models, complexity of data representation and non-determinism in data flow, time-related issues play crucial roles for assessing the composition correctness and requirements analysis.

In this work we present a framework for the formal representation, verification, and validation of the behavior of service compositions, specially focused on the specific issues of the Web services and SOA. This framework is based on the formal model suitable for the representation and reasoning on the behaviors of Web service compositions. This formal model provides a way to unambiguously represent the service specification and is exploited as a basis for the reasoning techniques and algorithms used in the analysis. This framework is implemented and incorporated in a toolkit that allows for various kinds of the automated verification and validation of business requirements, including search for communication-related, time-related, and data-related problems and their combinations.

Contents

Chapter 1

Introduction

In the past decades, the necessity for cross-enterprise application integration and interoperability, the simplification and reduction of costs of the integration and reconfiguration of the underlying applications, have been continuously driving the evolution of distributed computing. The Service-Oriented Architecture based on the Web service technology emerged as a natural consequence of this evolution. One of the key idea of this technology is the ability to create *service compositions* by combining and interacting with pre-exisiting services using standard languages and protocols, such as WSDL [W3C01] for the service interface definitions, and BPEL [ACD+03] or WS-CDL for the definition of the composition behavior.

The complexity of the composition model, its structure, and interactions between the involved services require continuous assessment of the possible behavior of the system-to-be in various settings considered by the analysts. The composite business applications often span across the enterprise boundaries, involving different stakeholders into the system modelling process. These stakeholders often come into play with their own business requirements, views and policies, which may be inconsistent or even contradictory under certain scenarios. The quality of the developed composition, the correctness of its behavior, and, as a result, the success of its procurement, strictly depend on the ability to ensure that the composition specification satisfies the desired business requirements of all the interested parties. Moreover, since the Web service technology is about loosely-coupled, autonomous applications, the main design questions concern the problems of who will use the service, in which way, and what is the expected outcome. The behavioral analysis is therefore required in order to understand the impact of the designed solution on the business domain and business policies prior the service is deployed and published.

In this settings, the approach to the design and development of the service-based distributed business processes should incorporate the formal modelling and analysis techniques that would allow to ensure the correctness of the behavior of the system [HBCS03]. In particular, the approach should be able to solve the following tasks:

- Formal modelling of the behavioral specifications of the compositions;

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

These tasks, however, were problematic to apply for the design of early distributed systems. The complexity of underlying technologies, tight coupling of the business requirements and the functional details required a solid expertize and involved a wide range of low-level issues, making the analysis process often unfeasible. On the contrary, the Web service technology, with its level of abstraction and standardization, was able to separate the business models from the low-level technicalities, and allows the analysis efforts to be performed directly at the level of business requirements and models.

Moving towards an analysis approach that supports the development of Web service composition, it is important to take into account the features peculiar to the Web service technology and the Service-Oriented Architecture, which require specific analysis approaches to model composition requirements, to search for the problems, and to encapsulate the analysis results into the system development process. Among them, the following aspects are of the topmost importance for the analysis.

- *Web service interactions*. The interactions with the Web services are essentially asynchronous – the send and receive actions are not synchronized by the partners. 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, however, considerably complicates the behavioral analysis of the Web service composition. Indeed, due to the complexity of queueing mechanisms and message overpasses, the behavior of the composition and the results of the analysis depend on the formal model adopted for the representation of the message delivery middleware, which is implementation dependent, and may vary in several ways (i.e., ordered/non-ordered queues, their number and structure). On the other hand, this problem is often ignored in most of the existing analysis approaches, thus often leading to incorrect results in verification of realistic scenarios.
- *Complexity of data model*. On 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. The necessity to manage data, however, brings the following problems to the composition analysis. First, the data domains are often infinite, and the semantics of data manipulating operations is complex . This restricts the applicability of traditional formal analysis techniques that rely on simple and finite system representations. Second, due to the distributed nature of the Web service applications and high level of abstractions implies that the internal details of the service implementations and data manipulation may be hidden to the analysts, making the proof of correctness more difficult.

- *Time-related issues*. In the analysis of Web service compositions it is required 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 this setting. 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 are committed to respect their own local timed constraints. 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. 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.

The main goal of this work is *to provide a framework for the formal representation, verification, and validation of the behavior of Web service compositions, specially focused on the issues specific to the Web service technology and Service-Oriented Architecture*. Such a framework would support the process of developing service compositions, providing a way to detect possible problems in the specification of the composition before the composition is being actually deployed.

The presented analysis framework is based on the formal model suitable for the representation and reasoning on the behaviors of Web service compositions. This formal model is given in two layers, where the first level aims at formalization of the Web service specification languages (i.e., BPEL/WSDL), and the second level provides a basis for the formal analysis techniques discussed here. In this model, the first layer is given in terms of process algebra notation, in particular LOTOS calculus. A two way mapping between BPEL/WSDL and LOTOS, thus enabling design and verification both in the Web service specification languages and in the presented formalism. An automated translation procedure may be used then in order to trace modifications between them. We remark, that the use of process algebra does not only provide a formal representation of the Web service specification languages, but also enables specific analysis techniques, such as bisimulation, hierarchical refinement, and service redundancy.

In the second layer, the composition is modelled as a finite set of service models that interact with each other through a formal model of the message delivery medium, namely *communication model*, used to represent the complexity of the service interactions that take place during the composition execution.

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

- *control flow* of the composition that defines the evolution of the composite system;
- *data flow* of the composition that defines the business data, its modification and exchange;

- *time flow* of the composition representing the timing properties of the model, such as duration of operations and time-related events.

In order to apply the formal analysis techniques based on the presented formalism, high-level composition specifications are translated into a low-level formal model. In particular, we define a mapping from the definition of the Web service interface in WSDL format [W3C01], and of the Web behavior in BPEL [ACD+03], into the underlying formalism. The representation of other composition languages, such as WS-CDL [W3C05], WSCI [W3C02], BPML [OMG05], and others may be analogously defined in a natural way.

The formal analysis techniques presented in this deliverable are based on *model checking* [CGP99], an analysis approach that, given a formal representation of the analyzed system and a formal definition of the behavioral requirements (e.g., deadlock freeness, a possibility to successfully complete the business transaction, guarantee that the data is managed correctly), performs an exhaustive exploration of all the behaviors of the system, generating a counterexample which represent the execution of the system that fails the desired 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.

We cope the challenges specific to the Web service domain, providing a set of approaches and algorithms, for modelling and analysis of these features. We address these challenges as follows.

In order to deal with the problem of modelling service interactions, we propose a model of composition, which is based on a parametric definition of the communication infrastructures. We define a hierarchy of communication models that are able to model a potentially infinite range of the composition scenarios [KP05, KPS06] and present an algorithm for the validation of 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. The outcome of the algorithm is then used to build the corresponding composition that can be used for further analysis.

Facing the complexity of the data in composition representations, we present an analysis approach that allows to represent and manage arbitrary data structures, and overcome the problem of the information incompleteness [KP06c]. The approach is based on the abstraction techniques [CGL94, GS97] that allow to take into account only those data-related 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.

We also extend the traditional quantitative analysis approaches with the ability to represent, verify and even compute timed characteristics of the Web service composition [KPP06b, KPP06a]. On the one hand, the formalism used to specify time flow in the behavior of the composition. 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, and exploit the duration calculus logic for the representation of complex timed requirements in the domain, such as timing of events, satisfaction of properties on subintervals, intervals sequencing, etc. Based on these formalizations, we provide techniques for the formal verification of the Web service composition with explicit representation of timed requirements. Finally, we introduce a decision procedure that can be used to compute extremal durations of intervals satisfying required (timed and non-timed) properties.

These analysis techniques and approaches are implemented and incorporated into a WS-VERIFY toolkit. The tool accepts the specifications of Web service compositions, given as a set of BPEL and WSDL documents, and translates them into the underlying formalism. Based on the translated model and the analysis settings provided by the analysts, the tool allows for various kinds of the verification and validation, including search for communication-related, time-related, and data-related problems and their combinations. For these purposes, the formal models are emitted in the form of the specifications accepted by a model checker, which performs the verification and provides the analysis results. Currently the tool supports two state of the art model checkers, namely NuSMV [CCGR00] and (partially) SPIN [Hol97]. Alternatively, the LOTOS models of the composition participants may be translated into a format accepted by the corresponding process algebra tools for the simulation-based types of analysis.

Chapter 2

Web Services and Service Compositions

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

Due to its intrinsic complexity, 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 [PTDL]. However, the SOA approach has to address these challenges taking into account features that are specific to SOA in order to deliver adequate and efficient solutions. Moreover, the ignorance of these features and the attempt to directly re-apply existing solutions potentially leads to a failure in the design and procurement of the service-oriented business applications.

2.1 Web Services

A Web service merely is a virtual software component accessible via multiple protocols. It has an interface described in a machine-processable format (specifically WSDL [W3C01]). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages [GRO00]. The service is published by the service provider, and may be looked up and discovered using Universal Description, Discovery, and Integration (UDDI, [OAS03]) standard.

The purpose of a WSDL document is twofold. First, it defines the abstract service interface, i.e., set of operations, type of their message parameters, and the definitions of the data types used in messages. Second, it defines the binding of this interface and its operations to a concrete network address, protocol, and message formats. More precisely,

the (abstract) Web service interface consists of the following elements:

- *data types* definition. User-defined data constructs and structures (for example, purchase order document definition, flight itinerary plan, customer records, etc) are defined here using XML Schema ([W3C04b]).
- *messages* definition. Messages are used to represent the input/output parameters of the service methods. They include user-defined or standard data types.
- *operations* definitions that specify the functionality provided by the service. The operations are logically grouped into *port types*. Each operation may potentially have one or more input parameters and may return an output message (or a specific fault message).

In this way, the WSDL document defines the Web service as a *stateless* computational entity. It says nothing about the internal state of the service, its evolution, and a behavioral protocol to be used in order to correctly use it. The Web service description should be augmented with some additional information that ranges from the ordering dependencies between methods to the partial definition of the control and/or data flow of the business process. Such behavioral interfaces may be defined using BPEL [ACD+03], Web service interfaces formalism [BCH05], or other models (see, e.g., [HNL02, vdA02, WM02, Bus03]).

2.2 Web Service Compositions

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 [KBG+04]. Integration and infrastructure management are the key elements for the architecture that uses services as basic constructs for easy composition of distributed business applications. These applications may be further published as services, thus creating a complex collaboration network, referred to as *service compositions* [KS02]. Such compositions are traditionally described using the terms *choreography* and *orchestration* [Pel03].

The term orchestration is used to describe a distributed process that is always controlled from the perspective of one of the composition participants. This participant is responsible for the design and implementation of the composed system. The orchestration often refers to an executable process that interacts with internal and external services accomplishing the goals of the orchestrator. Among a wide range of standards proposed for the definition of service orchestration ([Tha01, Ley01, OAS06], Business Process Execution Language (BPEL for short [ACD+03]) is probably the most used and industrially supported language; it is executable, and has rich notation allowing to express the majority of the business process and workflow modelling patterns [WvdADtH03].

On the contrary, the choreography defines the composition in a more anarchic way, defining a collaboration of independently created and controlled by separate agencies, where all the participants work and interact in order to accomplish a common goal. In such a collaboration there is no a single point of control over the execution of the process. The choreography is often referred to a composition specification, that is a blueprint of the composition behavior that the real composite applications should conform to. The WS-CDL (Web Service Choreography Description Language) continuously gains attraction due to its reach notation, formal background, and natural way to represent choreographies.

Regardless a particular viewpoint, the composition specification languages address the following requirements:

- The ability to represent various behavioral aspects of the composition: service *interactions* (i.e., message exchanges), *control flow* constraints (i.e., sequencing, conditional branching, iterative execution of application activities), *data flow* constraints (i.e., exchange, modification, evaluation of data and data expressions).

- The ability to handle different execution modes: apart from normal flow of the execution, the language is aimed at representing exceptional and transactional behavior, providing facilities to coordinate, recover, and compensate abnormal behavior of the system. The problem of transactional support is particularly important, since in the distributed and loosely-coupled context the classical transactional mechanisms fail [Tyg96]; there is a need to deal with Long-Running Transactions (LRT), where business activities may last for days, weeks or even months. 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. This requires modularization, abstraction, coarse granularity in the representation of activities. As a consequence, the ability to express *non-determinism* becomes essential in order to unreveal certain implementation details, model a freedom of choice, or to abstract from a certain application logic.

The complexity of the composition model, the necessity to integrate independent services, potential non-determinism and incompleteness require a rigorous and accurate correctness analysis of the composition behavior.

2.3 Analysis of Web Service Compositions

The analysis of Web service composition aims at checking its correctness with respect to a certain set of functional and non-functional requirements. These requirements express various composition properties, ranging from high-level strategic business goals

and needs, to low-level technical constraints and policies. Apart from the vertical classification, Web service composition requirements may be classified horizontally: here global requirements express the business properties of the composition as a whole, while local requirements express the expectations, offers, and obligations of a participant to a composition [W3C04a]. In these settings, the problem of understanding, modelling, unrevealing, and analyzing of requirements becomes a cornerstone problem for the composition design [CMS05, OYP05], requiring integration of business requirements models in the business process life cycle [KPR04, AJKL06]. These business requirements may be used in the following ways:

- Requirements that represent *required* properties of the composite system (assertions). These 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.
- Requirements that represent *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 possibility property is expected to be satisfied in some execution scenario of the composite system.
- Requirements that model *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. 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).

2.3.1 Forms of Correctness Analysis

The analysis of Web service composition may be conducted in different ways. Its usage also depends on the applied composition development strategy.

In the top-down approach (i.e., choreography-driven), an abstract specification of the composition is being designed. This specification is later populated with real service implementations. In this case, it is important to verify that the abstract model is able to satisfy both the required and desired properties. It is also necessary to identify and extract the set of obligations (assumed properties) that are necessary to guarantee the correctness of the future composition implementation. In the next steps, when the composition specification is being populated with the real services, the analysis is applied as follows. First, it is necessary to check that the implementation conforms to the abstract specification (conformance testing [GP03]). The specification may be iteratively refined, providing more and more detailed composition model. At each step of the refinement it is necessary to guarantee that the refined model satisfies all the relevant properties of the prototype.

Second, it is necessary to verify that the implementation satisfy the assumptions posed on the abstract participants during the design of the composition specification.

In the bottom-up development approach the composition is being constructed from the specification of existing services, resulting in a executable distributed application. In this case each local service specification may be defined as a BPEL process. The analysis applied for such a strategy includes the composability/compatibility analysis [CPT01, BCPV04], the verification of the business goals and requirements (both required and desired properties) over 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. It is important here to be able to verify that one of the service may be replaced by another without affecting the composition properties.

2.3.2 Challenges and Problems

The correctness analysis of service-based applications and compositions should tackle certain problems that are specific to the service-oriented architecture and the Web service technology.

First, the services are loosely-coupled, they may be designed and implemented autonomously by independent parties, and their invocation and usage may be costly and/or time-consuming. This makes testing of composite applications rather problematic if at all possible. Often the only basis for the composition analysis is the set of service specifications and descriptions, such as interfaces, behavioral descriptions, contracts, etc. In these settings, the static verification and simulation techniques are more appropriate for the composition analysis at design-time, while the correspondence between the specified and actual service behavior may be checked at run-time using various monitoring techniques [FF95, SMvMC02, BTPT06].

Second, service compositions are defined using high-level description and specification languages, that allow to focus on the business logic of the system and to abstract from the low-level implementation details, such as middleware, implementation languages, etc. While irrelevant from the business logic point of view, these details may be critical for the composition correctness since they may affect the ways the system operates. This is a case, e.g., for the implementation of communication models (i.e., queuing systems), since the way the messages are stored, exchanged and ordered at the middleware level may change the behavior of the system at the process level.

Third, the service descriptions and specifications often contain only partial information needed to perform correctness analysis. Indeed, the details of an actual service implementation are not revealed to the consumers, only some necessary descriptions are provided (e.g., interfaces, preconditions and effects, conversation specifications, policies).

This makes the non-determinism and incompleteness an inherent property of the service composition models, that requires specific the analysis techniques and approaches, different from those applied to fully specified software systems. As a consequence, the analysis of different assumptions and constraints over such an incomplete specifications becomes crucial for ensuring the composition correctness.

Last, but not least, in the analysis of the composition it is important to take into account also different non-functional, quantitative requirements, e.g., cost, time, resource consumption. Time properties are particularly relevant in this context. Indeed, for a wide range of applications that deal with long-running activities and transactions, a composition is expected to fulfill various timed constraints posed by interested parties. Failure to satisfy these constraints leads to alternative behaviours controlled, e.g., by timeouts, and therefore the functional aspects of the composition behavior are deviated by non-functional properties.

Considering these problems, we see the service compositions analysis approach as a framework that satisfy the following requirements:

- allow for static verification of Web service composition specifications. The analysis is done before the actual deployment/execution of the composite system, and allows to detect potential flaws and check that various requirements and properties are satisfied by the specification.
- provide a set of techniques for formal verification and validation of the compositions, taking into account control, data, and time flow of the composite system. This requires an expressive formalism for the composition representation, for modelling various requirements (in particular time properties), and efficient reasoning techniques.
- provide a way to deal with the incompleteness of the specification models, and to take into account the influence of the abstracted details on the behavior of the composition. This requires a methodological support for the extraction of obligations and constraints over the unknown functionality, and for injecting these assumptions into the analysis process.

Chapter 3

Formalization of Web Service Composition

In this chapter we present a formal model for a generic Web service composition model that allows for the representation of various aspects of the composition behavior. We also present a formal specification language, namely Linear-time Temporal Logic that is exploited for the formal specification of functional requirements and constraints of the composition models. These formalizations are later used as a basis for the formal analysis approach applied to the composed business processes.

3.1 Composition Architecture

A Web service composition may be represented in the following generic way (Fig. 3.1). The composition describes a collaborative behavior of several interacting participants, potentially including both users and applications. These participants are represented with their descriptions that include interface, policies, etc. The behavior of a service participating to the composition may be given as a local process specification, in particular, using BPEL specification. The choice of BPEL as a language for the definition of Web service behavior is dictated by the following reasons. First, BPEL is a widely accepted standard being developed by OASIS [OAS93], and widely supported by industry implementations. Second, BPEL is one of the most expressive languages for the description of the composition behavior [WvdADtH03]. It provides a highly expressive notations for the representation, e.g., of 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 [W3C05], WSCI [W3C02], BPML [OMG05], and others.

Performing their activities, the participants exchange messages using a complex communication medium implemented by underlying middleware systems. This medium is

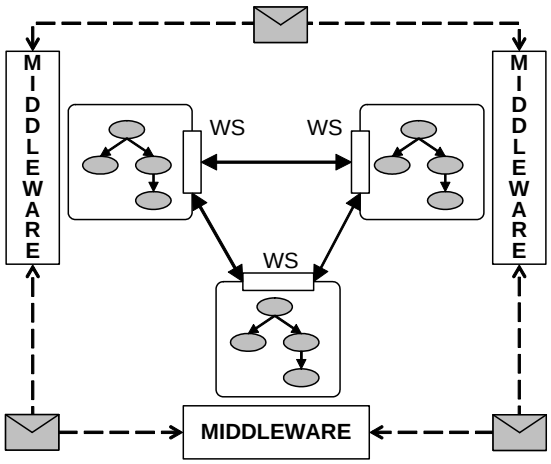

Figure 3.1: Generic Web service composition

responsible for message transmission, delivery, and queuing. A concrete implementation of this medium is abstracted away from the composition specification and may vary both for composition instances and participating service implementations.

The composition model relies on certain hypotheses on the Web service compositions. These hypotheses define some boundaries to the scope of our verification framework, and will allow us to abstract from low-level issues that are irrelevant for the "logic" of the composition, and to simplify the formalization of the model:

- The communication channels used by the participants are disjoint. That is, it is never the case that two processes are able to invoke or receive the same operation from a third process. Formally, this means that the sets of communication actions used by pairs of processes are disjoint, and that the end points of the communications are statically fixed.

- The communication channels are perfect, i.e. no message losses can ever occur. This assumption may be enforced by special techniques in the WS world. In particular, WS Reliable Messaging or monitoring techniques can be used for these purposes.

- The number of processes participating to a composition is fixed, the processes cannot be created or destroyed dynamically. Formally, this means that the composition model is fixed.

- While the operations performed by the participants to the composition are durable in general, 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 their tasks

and take their decisions. The "technical" times, which are required, for instance, by the communications among BPEL processes and by the BPEL engines to manage incoming and outgoing message, are orders of magnitude smaller and can be neglected.

Under these assumptions, we consider a very general communication infrastructure that allows for modeling the following features:

- 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.)

- An arbitrary implementation of the underlying queuing mechanism is allowed. That is, we do not commit to a specific model of implementation of the queues. In order to reflect this property, we assume the most general behavior of queue, 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 every specific engine.

3.2 Web Service Model as a LOTOS Process

We formalize the BPEL/WSDL Web service specification using LOTOS [ISO89], one of the most expressive process algebra. A two-way mapping between BPEL/WSDL and LOTOS is presented, together with general guidelines for translations between BPEL/WSDL and a process algebra. We choose LOTOS because it allows us the data handling, and the verification and the modelling of the BPEL handlers. See [Fer04] for more details on the translation.

3.2.1 LOTOS in a Nutshell

LOTOS is a specification language for distributed open systems normalized by the ISO. It combines two specification models: one for static aspects (data and operations) which relies on the algebraic specification language ACT ONE [EM85] and one for dynamic

aspects (processes) which draws its inspiration from the CCS [Mil89] and CSP [Hoa84] PAs.

Basic LOTOS

This PA authorizes the description of dynamic behaviors evolving in parallel and synchronizing using rendezvous (all the processes involved in the synchronization should be ready to evolve simultaneously along the same action). A process P denotes a succession of *actions*, which are basic entities representing dynamic evolutions of processes; a process can be recursive. The symbol **stop** denotes an inactive behavior (it could be viewed as the end of a behavior) and the **exit** one depicts a normal termination. The specific τ action corresponds to an internal evolution.

Now, we present LOTOS behavioral operators. The prefixing operator G**;**B proposes a sequence. The non-deterministic choice between two behaviors is represented using **[]**, and LOTOS parallel composition operator is given by the expression B_1 $|[G_1, ..., G_n]|$ B_2 expressing the parallel execution between behaviors $B1$ and $B2$. It means that $B1$ and $B2$ evolve independently except on the actions G_1, ..., G_n on which they evolve at the same time firing the same action (they also synchronize on the termination **exit**). The disabling operator $B_1[>B_2$ model the interruption: the behavior B_1 could be interrupted at any moment by the behavior B_2; when B_1 is interrupted, B_2 is executed (without having interruptions).

Full LOTOS

Data expressions and value passing is expressed in LOTOS as follows. A process is parameterized by a (optional) list of formal actions $G_{i\in 1..m}$ and a (optional) list of formal parameters $X_{j\in 1..n}$ of type $T_{j\in 1..n}$. The full syntax of a process is the following:

process P **[**G_0, ..., G_m**]** (X_0:T_0, ..., X_n:T_n) : $func$:= B **endproc**

where B is the behavior of the process P and $func$ corresponds to the functionality of the process: either the process loops endlessly (**noexit**), or it terminates (**exit**) possibly returning results of type $T_{j\in 1..n}$ (**exit**(T_0, ..., T_n)).

Action identifiers are possibly enhanced with a set of parameters (offers). An *offer* has either the form G**!**V and corresponds to producing a value V, or the form G**?**X:S which means the assignment of a value of type S to a variable X. A behavior may depend on Boolean conditions. Thereby, it is possible that it be preceded by a guard **[***Boolean expression***]** $\rightarrow B$. The behavior B is executed only if the condition is true. In the sequential composition operator, the left-hand side process can transmit some values (**exit**) to a process B (**accept**):

... **exit**(X_0, ..., X_n) $\gg$ **accept** Y_0:S_0, ..., Y_n:S_n **in** B

3.2.2 The two-way mapping between LOTOS and BPEL

In this section we show the two-way mapping between LOTOS, a process algebra that allows data handling, and BPEL. Our goal is showing a two-way mapping between the two languages, that allows an automated translation. The mapping is represented in Table 3.1.

Basic behaviors and interactions

At the core of BPEL process model is the notion of peer-to-peer interaction between partners described in WSDL. An interaction is characterized by the partner link, the port type, and the operation involved in the two communicating partners (each partner defines these three elements for each interaction). When the process representing a service is defined, an action is simply an emission or a reception.

The reception of a message is expressed using the *receive* activity in BPEL and using a action with a reception in all its parameters in LOTOS.

In BPEL, the emission is written with the *reply* or the *asynchronous invoke* activity whereas in LOTOS we use a action with an emission in all its parameters. The BPEL *synchronous invoke*, performing two interactions (sending a request and receiving a response) corresponds in LOTOS to an emission followed immediately by a reception. In LOTOS we have two different actions, because we have two interactions in BPEL.

Structured Behaviors

The mapping for LOTOS dynamic constructs and BPEL structured activities is defined as follows. The *pick* BPEL activity is executed when it receives one message defined in one of its *onMessage* tag or when it is fired by an *onAlarm* event. The equivalent construct in LOTOS is obtained using the non deterministic choice, in which the first action of each branch is a reception; the branch whose beginning reception is performed first is chosen.

The *sequence* activity in BPEL match with the LOTOS prefixing operator ';'. BPEL *flow* activity is modelled in LOTOS with the full synchronization constructs '||'. The mapping of the *link* tag is more complicated, since LOTOS does not have an explicit dependence relation between concurrent actions. In BPEL we specify with the *source* tag the activity that has to occur first, and with the *target* tag the dependent activity. In LOTOS we have an action for each link. These actions are put after the end of the source behavior, and before the beginning of the target one; the two behaviors synchronizes on these actions, that is they have to execute them at the same time. In this way we are sure that the source behavior is completed before the beginning of the target one.

The *switch* tag defines an ordered list of *case* tag. A case corresponds to a possible activity which may be executed. The condition of a case is a Boolean expression on

Sample BPEL Code	Sample LOTOS Specification
<assign ... > <copy> <from expression="5"/> <to var="x"/> <copy> </assign>	exit(5) ≫ accept x
<receive ... variable="m"> </receive	g?m;
<reply ... variable="m"> </reply>	g!m;
<invoke ... invar="mS" outvar="mR"> </invoke>	gS!mS; gR?mR;
<pick ... > <onMessage ... variable="m1"> < ... act1 ...> </onMessage> <onMessage ... variable="m2"> < ... act2 ... > </onMessage> </pick>	(g1?m1; ..act1..) [] (g2?m2; ..act2..)
<sequence ...> < ... act1 ... > < ... act2 ... > </sequence>	..act1..; ..act2..
<flow ... > < ... act1 ... > <source linkname="link1" condition="cond1"/> </act1> < ... act2 ... > <target linkname="link1"/> </act2> </flow>	..act1..; ([cond1]->link1 !1; [] [not(cond1)]->link1 !0;) || (link1 ?x:Bool; ([x=1]->..act2.. [] [x=0]->i;))
<switch> <case condition= "bpws:getVariableData(x)>=0"> <..act1..> </..act1..> </case> <otherwise> <..act2..> </..act2..> </otherwise> </switch>	[x>=0] -> ..act1..; [] [x<0] -> ..act2.. ;
<while condition= "bpws:getVariableData(x)>=0"> <..act1..> </..act1..> </while>	proc while1 [..](..) := [x<0]-> i; [] [x>=0]->..act1..; while1[..](..) endproc

Table 3.1: The BPEL-LOTOS two-way mapping examples

variables. In our process algebra we have a standard pattern combining guarded expression and non deterministic choice, very often used in the design with LOTOS. The *while* BPEL tag and LOTOS recursive processes correspond to each other. The condition of the *while* is the exit condition of the recursive process. The behavior of this recursive process matches exactly the body of the BPEL loop, and conversely. The recursive process is instantiated by the process corresponding to the scope that contains the while. In the LOTOS modelling, recursive processes have to respect the structure of the BPEL while, in order to simplify the translation.

3.2.3 Data Manipulation

LOTOS allows for the definition of abstract data types, that is data domains and operations on them (e.g. a list with operations: add an element, extract the first one etc.); many basic types (char, natural, etc.) are already defined. In BPEL, types are described using XMLSchema; elements can be simple (lots are already defined) or complex (composed by other elements). A simple element and LOTOS basic data type corresponds each other. A complex element in XMLSchema may be modelled using LOTOS abstract data types and data structures.

In LOTOS, only process parameters need to be declared (not necessary for action variables) whereas in BPEL either global and local variables involved in interactions have to be declared. In LOTOS, in local process we can declare local variables. A BPEL *message* corresponds to a set of action parameters in LOTOS. In particular a BPEL *part* corresponds to a parameter of an action in LOTOS.

The BPEL *assign* tag has three equivalents in LOTOS depending on their use: (i) *let* X_i*:*T_i*=*V_i *in B* means the initialization of variables X_i of types T_i with values V_i ($\forall i \in 1..n$) in the behavior B, (ii) B_1*; exit(*Y_i*)* $\gg$ *accept* X_i*:*T_i *in* B_2 denotes the modification of variables X_i (replaced by new values Y_i), (iii) ***P(***X_i***)*** is an instantiation of a process or a recursive call meaning assignments of values X_i to the parameters of the process P. Conversely, these LOTOS constructs can be mapped into BPEL using *assign*, and more precisely the *copy* tag.

BPEL scope and LOTOS process pattern

In BPEL the *scope* tag defines a behavior context (local variables, event handlers, fault handlers, compensation handler) for its primary activity. The primary activity describes the normal behavior of the scope. In LOTOS we can define a pattern of processes that behaves in the same way. We point out that a LOTOS user, in the design of a scope with handlers have to respect this pattern of processes, in order to obtain automatically BPEL code. Vice versa, from BPEL specification we can get the LOTOS one, automatically filling this pattern of processes.

In BPEL the scope can be nested. The outermost scope is the global BPEL process.

In LOTOS we have the concept of local process. In LOTOS the process/scope is local to the outer process/scope. Each process/scope instantiate the following processes:

- *primary activity*: a process $primaryActivity$ for the primary activity of the *scope*. In the case of normal termination, its last action is an end.
- *event handler*: a process $eventHandlers$, executed in parallel with its primary activity.
- *fault handlers*: a process $faultManager$ that catches a fault launches the process $Kill$ to terminate the $primaryActivity$ and $eventHandlers$, then, depending on the fault name, calls the corresponding subprocess to perform the fault activities.
- *compensation handler*: a process for the compensation handler. In BPEL we can have at most one compensation handler in a *scope*. The name of the compensation handler is the name given in the *scope* attribute of the activity *compensate*.

Each process/scope has the following structure (LOTOS pseudo-code):

```
proc scopeName [..](..) :=
 ( (primaryActivity[..](..) || eventHandlers[..](..))
    [> Kill[]()
 )
 |[fault,end]| faultManager[..](..)
endproc
```

The $eventHandlers$ is concurrent with $primaryActivity$; they both can be interrupted by the process $Kill$, launched by the $faultManager$ when a fault occurs. The process for the compensation handler is called inside a process representing a fault or another compensation handler: in BPEL it can be invoked, using the *compensate* tag, only either in a fault handler or in another compensation handler.

Fault Handlers When a fault occurs in a BPEL scope, all activities in the primary activity and in the event handlers of the scope begin to terminate. In LOTOS we define a process *faultManager* running concurrently respect the process representing the scope, synchronizing on the actions *fault* and *end*. The *fault* action has the parameter $faultName$ to communicate the name of the fault; the *end* does not have parameters because we do not need to send or to receive messages, but only to communicate an event. The faultManager definition is the following:

```
proc faultManager [fault, end] (faultName:String):=
  ( fault?faultName:String;  Kill;
    [faultName1]-> faultProc1[..](..)
```

```
    [faultName2]-> faultProc2[..](..)
    ..
  )
  [] end;
endproc
```

If the scope terminates without faults, the process representing the scope performs as last action the action end, allowing to $faultManager$ to terminate without doing nothing. A fault in BPEL is launched through the tag *throw* (that has with attribute the name of the fault) or as response to an invoke activity; in LOTOS through action $fault$. After this the process $Kill$ is instantiated. This process doing nothing, but terminate $primaryActivity$ and $eventHandlers$ using the disabling operator '$[>$'. Finally, the process corresponding to the fault name is chosen: for example $faultProc1$ corresponds to the fault $faultName1$.

We consider now the problem of fault propagation and handling. In BPEL, when a fault occurs in a scope S that cannot handle it, S terminates abnormally and the fault is propagated to the next scope up. If S can handle the fault, it terminates normally after executing the fault handler activities. From BPEL to LOTOS translation we know which fault handler will catch a fault by parsing the BPEL files. Similarly, from LOTOS to BPEL translation, by parsing the LOTOS specification we know the fault handler that will catch the fault; if the fault is not caught, we have a *stop* action instead of the *fault* one.

Compensation Handlers While a business process is running, it might be necessary to undo one of the steps that have already been successfully completed. To each scope we can optionally associate its compensation handler that undoes the primary activity of the scope; once a scope completes successfully, its compensation handler become ready to run. This can happen in either of two cases: explicit or implicit compensation. We map the compensation handler into a LOTOS process local to the process representing the scope.

explicit compensation: It occurs upon the execution of a *compensate* activity, that can occur inside a fault handler or a compensation handler of the scope immediately enclosing the scope to be compensated; the *compensate* activity has an attribute *scope* whose value specifies the name of the scope to be compensated. The *compensate* activity is modelled in LOTOS by a call to the process representing the compensation handler associated with the scope.

implicit compensation: It occurs when there is a fault handling. Let A be a scope, and B an its nested compensable scope. Consider the following scenario: B is completed successfully, but another activity in A throws a fault. Implicit compensation ensures that whatever happened in scope B get undone by running its compensation

handler. Therefore, the implicit compensation of a scope goes through all its nested scopes and runs their compensation handlers in reverse order of completion of those scopes. We can map this mechanism in LOTOS by calling in the same order the processes representing the compensation handlers; all these calls are executed in $faultManager$ of A before the beginning of the fault activities. Obviously we have to store the order in which the scopes are completed. We can use a queue data structure in LOTOS to do it; this queue has global visibility and it is updated when a scope completes or if a scope is compensated. The process $faultManager$ of A can use this structure to know the order of completion.

Event Handlers We have to consider the BPEL semantics of the event handler: it can accept messages an arbitrary number of times, until the scope ends. We adopt in LOTOS a recursive process, concurrent to the primary activity of the scope in which is contained. The structure of $eventHandlers$ is defined as follows:

```
proc eventHandlers [onMessage1, onMessage2,..](..):= ( (
(onMessage1?m1:T; ..act_m1..;) [] (onMessage2?m2:T;
..act_m2..;) [] .. ) eventHandlers [onMessage1,
onMessage2,..](..); ) [] end; endproc
```

The action *onMessage1* represents the reception of a message $m1$, whose type is T. After receiving the message, the corresponding activity *act_m1* is executed. Then the process recursively calls itself, and it ends when an *end* interaction happens.

3.3 Web Service as State Transition System

While the LOTOS specification provides a mapping from the high-level description of a Web service specification, the State Transition System formalism described below and the notion of the STS composition is exploited in this work as a basis for the formal verification and validation of the distributed service-based applications. This formal model 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 processes 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 processes. The control flow is defined by the behavioral model, used to represent a behavior of the service specified in BPEL or LOTOS process. We encode processes as *state transition systems*, which describe 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 process states.

3.3.1 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 $\langle \mathcal{T}, \mathcal{V}, \mathcal{F}, \mathcal{I}_f \rangle$, where

- *$\mathcal{T}$ is a set of (possibly infinite) types;*
- *$\mathcal{V}$ is a set of typed variables;*
- *$\mathcal{F}$ is a set of typed functions;*
- *$\mathcal{I}_f$ is the function that given a typed function $f \in \mathcal{F}$ and a set of values $v_1, \ldots, v_n$ returns the result of the computation of $f(v_1, \ldots, v_n)$.*

As one can see, this model is generic enough to define the management of the data in arbitrary Web service composition represented as a set of interacting processes. Indeed, it allows to define arbitrary data types, variables, XPath expressions and predicates, represented above.

A ground state is characterized by a complete assignment over the set of typed variables $\mathcal{V}$.

Definition 3.2 (Ground State) *Given a ground context $\langle \mathcal{T}, \mathcal{V}, \mathcal{F}, \mathcal{I}_f \rangle$, we define a* ground state *g as a set of pairs $\langle x, v \rangle$ such that for all $x \in \mathcal{V}$ there exists a unique value v belonging to the type of x.*

Let e denote an expression and E a set of expressions. We use t to denote a term, and T to denote the set of terms. We also write x for a variable in $\mathcal{V}$. The syntax of expression is as follows[1]:

- $E \equiv (t_1 = t_2) \mid \neg e \mid (e_1 \vee e_2)$ that is equality between terms, negation or disjunction of expressions;
- $T \equiv x \mid f(t_1, \ldots, t_n)$ that is a variable or function call on terms.

A *condition* $\phi \in \Phi_D$ is an expression of the form presented above. An *assignment* $\omega \in \Omega$ has the form $(x := t)$.

[1]We remark that in this model the constants may be represented as functions with zero parameters. Analogously, boolean variables and predicates may be also interpreted as abstract variables and functions.

Definition 3.3 (Evaluation Function) *We define the* evaluation function Γ_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*:*

- $\Gamma_g(x) = v$*, where* x *is a variable and* $\langle x, v\rangle \in g$*;*
- $\Gamma_g(f(v_1, \ldots, v_n)) = v$*, where* $v = \mathcal{I}_f(f, v_1, \ldots, v_n)$*;*
- $\Gamma_g(f(t_1, \ldots, t_n)) = \Gamma_g(f(\Gamma_g(t_1), \ldots, \Gamma_g(t_n)))$*;*
- $\Gamma_g(t_1 = t_2) = true$ *iff* $\Gamma_g(t_1) = \Gamma_g(t_2)$*;*
- $\Gamma_g(\neg e) = \neg\Gamma_g(e)$*;*
- $\Gamma_g(e_1 \vee e_2) = \Gamma_g(e_1) \vee \Gamma_g(e_2)$.

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* $update(g, \omega)$*, is the state* g' *where* $\langle x', v\rangle \in g'$ *if* $\langle x', v\rangle \in g$ *and* $x \neq x'$*, or if* $x = x'$ *and* $v = \Gamma_g(t)$.

3.3.2 Time Model

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

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 $true \mid c \sim t \mid \phi_1 \wedge \phi_2$, where $\sim \in \{\leq, <, =, \geq, >\}$, 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.

3.3.3 STS Formalism

A state transition system (STS) defines the behavior of the process. STS has a finite number of states, or *program counters*, that specify execution points of the process. The

execution of the STS is given by the transition relation. The transition may be executed in a state s, if it is the transition *guard* evaluates to true in this state. When the transition is executed, an action is fired, and a set of assignments is performed.

Following the standard approach in process algebras, we distinguish *external actions* representing process interactions, and *internal actions* Θ, which are used to represent evolutions of the system that do note involve interactions with the external services. External actions are distinguished in *input actions* $\mathcal{I}$, which represent the reception of message α, and *output actions* $\mathcal{O}$, which represent messages α sent to external services. The message is identified by the operation (message type) $\mu(\alpha)$ and its content $\text{VAL}(\alpha)$. We denote a set of all messages as M. The send action is denoted as $\overrightarrow{\mu}(\bar{x})$, where $\bar{x}$ is a vector of process 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 $\bar{x}$. Set of all actions is denoted as $\mathcal{A}$.

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 states may be equipped with the time *invariants* that constrain the time that may be spent in the state. Finally, a set of clocks may be reset when a particular state is entered.

Definition 3.5 (STS)
A state transition system *is a tuple* $\langle \mathcal{S}, \mathcal{S}_0, \mathcal{V}, \mathcal{X}, \mathcal{A}, \mathcal{R}, \mathcal{L}_I \rangle$*, where*

- $\mathcal{S}$ *is the set of states and* $\mathcal{S}_0 \subseteq \mathcal{S}$ *is the set of initial states;*
- $\mathcal{V}$ *is a set of process variables and* $\mathcal{X}$ *is a set of process clocks;*
- $\mathcal{A}$ *is a set of process actions;*
- $\mathcal{R} \subseteq \mathcal{S} \times \Phi \times \mathcal{A} \times \Omega^* \times 2^{\mathcal{X}} \times \{true, false\} \times \mathcal{S}$ *is the transition relation;*
- $\mathcal{L}_I : \mathcal{S} \to \Phi_T$ *are the functions assigning time invariants to states.*

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

3.4 Web Service Composition Model

We now give a formal model and semantics of the Web service composition. In this model the composition is represented as a network of STS corresponding to the participating

processes 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.

3.4.1 Modelling Message Interactions

In order to represent a composition of Web services, we now define *state transition systems with channels*. This model describes the executions of the composition according to a parametric definition of the communication model.

Intuitively, this model is characterized by a set of global states, describing the composition during its execution, and a set of (ordered or unordered) queues that store the messages exchanges among partners. A global state contains two components: a *control state* that represents the local states of the participating STSs, and a *queue content* that defines sets of messages stored in the queues in a particular moment of time.

More precisely, let us assume that the composition is built from n STSs representing the participating Web services. We represent a *global program counter* as a vector $\bar{s} = \langle s_1, \ldots, s_n \rangle$, where s_i is a local state (program counter) of the i^{th} STS. We denote a vector with component s_i updated to s'_i as $\bar{s}[s_i/s'_i]$. Let us also assume to model the communications among Web services with set of $m > 0$ queues with disjoint alphabets $M_j \subseteq M$. 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 composition is a tuple* $\Delta = \langle B, \mathcal{L}_M, \mathcal{L}_O \rangle$*, where* $B = \langle b_1, \ldots, b_m \rangle$*, is a vector of queue bounds,* $\mathcal{L}_M : M \rightarrow [1 \ldots m]$ *is a function that associates a message* α *with a queue* i*, and* $\mathcal{L}_O : [1 \ldots m] \rightarrow \{\top, \bot\}$ *is a function that declares the queue as either ordered or unordered. The alphabet* M_i *of queue* i *is defined as* $M_i = \{\alpha \mid \mathcal{L}_M(\alpha) = i\}$*.*

Let M^* be a set of sequences (or strings) of elements from M. Let also $\mathbb{N}^M$ be a set of multisets of M, i.e. set of mappings from M to natural number $\mathbb{N}$. Given two elements w and w', we write $w.w'$ to denote the string concatenation, if $w, w' \in M^*$, and multiset union, if $w, w' \in \mathbb{N}^M$. We 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': $\forall\, \alpha \in M, w(\alpha) \leq w'(\alpha)$.

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 values of the corresponding queue variables $\mathcal{V}^q$.

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 $w_j \in \mathbb{N}^M$ and j^{th} queue is unordered. We extend the 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. Analogously, $C \leq C'$ if for any queue index i $w_i \leq w'_i$. We write $|C| \leq B$ to specify that $|q_i| \leq b_i$.

The queue content represents the values of the stored messages. When an output action $\overrightarrow{\mu}(\bar{x})$ is performed, the corresponding values are assigned to the queue variables that represent the content of the message: $\bar{x}^q := \bar{x}$, where $\bar{x}^q \in \mathcal{V}^q$. Analogously, when the message is consumed (action $\overleftarrow{\mu}(\bar{x})$ is fired), the process variables get their values from the content of the message: $\bar{x} := \bar{x}^q$.

Definition 3.7 (State Transition System with Channels)
A State Transition System with Channels *(CSTS) for the composition of n STSs under a communication model $\Delta = \langle B, \mathcal{L}_M, \mathcal{L}_O \rangle$ is a transition system $\langle GS, GS_0, \mathcal{V}, \mathcal{X}, \mathcal{A}, T, \mathcal{L}_I \rangle$ where*

- *GS is a set of* global states *of the form $\langle \bar{s}, C \rangle$, and GS_0 is a set of initial global states with $C = \langle \epsilon, \ldots, \epsilon \rangle$;*
- *$\mathcal{V} = \mathcal{V}^q \cup \bigcup_i \mathcal{V}^i$ is a set of all local and queue variables, and $\mathcal{X} = \bigcup_i \mathcal{V}^i$;*
- *$\mathcal{A} = \bigcup_i \mathcal{A}^i$ is a set of actions;*
- *$T \subseteq GS \times \Phi \times \mathcal{A} \times \Omega^* \times 2^{\mathcal{X}} \times \{true, false\} \times GS$ is the global transition relation. A transition $t = (\langle \bar{s}, C \rangle, \phi, a, \Omega, Y, \vartheta, \langle \bar{s}', C' \rangle)$ is in T, if for some $1 \leq i \leq n$, $\bar{s}' = \bar{s}[s_i/s'_i]$ and $(s_i, \phi, a, \Omega_i, Y, \vartheta, s'_i) \in \mathcal{R}^i$, and one of the following holds:*
  - *$a = \overrightarrow{\mu}(\bar{x}) \ \wedge \ C(gs') = C(gs).\alpha \ \wedge \ |C(gs')| \leq B \ \wedge \ \mu(\alpha) = \mu \ \wedge \ \Omega = \Omega_i \cup (\bar{x}^q := \bar{x})$;*
  - *$a = \overleftarrow{\mu}(\bar{x}) \ \wedge \ C(gs) = \alpha.C(gs') \ \wedge \ \mu(\alpha) = \mu \ \wedge \ \Omega = \Omega_i \cup (\bar{x} := \bar{x}^q)$;*
  - *$a \in \Theta \ \wedge \ C(gs') = C(gs) \ \wedge \ \Omega = \Omega_i$.*
- *$\mathcal{L}_I : GS \to \Phi_T$ is a functions that assigns invariants to the global states, such that $\mathcal{L}_I(\bar{s}) = \bigwedge_i \mathcal{L}_I^i(s_i)$*

When an internal action is performed the state of the queues is not changed. When an external transition is performed, the data is passed to and from the queues. That is, for message output the message is added to the queue with the content populated from the corresponding local variables of the emitting process. Indeed, the output transition is allowed if the corresponding queue is not full. Analogously, for the input actions the local variables are assignment from the message content. The transition is possible if there is a corresponding message stored in the queue.

3.4.2 Semantics of Web Service Composition

The evolution of the composition is defined in terms of data, time, and message contexts. During the execution of the composition variables are modified, time increments, and messages are added and removed from the queues. The point of the composition executions is represented as a *configuration* of the composition. A configuration γ is a tuple $\langle \bar{s}, \bar{g}, C, \bar{u} \rangle$, where $\bar{s} = \langle s_1, \ldots, s_n \rangle$ is a *global program counter*, $\bar{g} = \langle g_1, \ldots, g_n, g^q \rangle$ is a *global ground state*, C represents a queue structure, and $\bar{u} = \langle u_1, \ldots, u_n \rangle$ is a *global clock valuation*.

Data flow of the composition is managed by the assignments and data conditions on transitions. That is, the assignment defines the modification of the global ground state in the current configuration. A transition is possible only if the transition guard evaluates to true in the source configuration of the transition.

Timed behavior of the compositions is based on the formalism of *timed automata*. In particular, the fact that the operation takes 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. In order to guarantee that the transition will take place at the right moment of time, the states and transitions are annotated with the invariants and guards of the special clock variables. The state invariants should be true when the system is in the state. More formally, the above model is extended as follows.

The semantics of the web service composition is defined as *global transition system* (GTS). We will write $\langle \bar{s}, \bar{g}, C, \bar{u} \rangle \models \phi$ to denote that the configuration satisfies ϕ.

Definition 3.8 (Global Transition System)
The semantics of a CSTS $\langle GS, GS_0, \mathcal{V}, \mathcal{X}, \mathcal{A}, T, \mathcal{L}_I \rangle$ is defined as a global transition system $\Sigma_\Delta = (\Gamma, \Gamma_0, \rightsquigarrow)$*, where Γ is a set of configurations of the form $\langle \bar{s}, \bar{g}, C, \bar{u} \rangle$, $\Gamma_0 \subseteq \Gamma$ is a set of initial configurations with $\bar{u} = \langle u_{10}, \ldots, u_{n0} \rangle$, and $\rightsquigarrow \subseteq \Gamma \times \{\mathcal{A} \cup \text{TICK}\} \times \Gamma$ is a transition relation such that:*

- $(\langle \bar{s}, \bar{g}, C, \bar{u} \rangle, \text{TICK}, \langle \bar{s}, \bar{g}, C, \bar{u} + d \rangle) \in \rightsquigarrow$*, if*
  - $\forall\, (\langle \bar{s}, C \rangle, \phi, a, \Omega, Y, \vartheta, \langle \bar{s}', C' \rangle) \in T$*, if* $\vartheta = true$ *then* $\bar{g}, \bar{u} \not\models \phi$*;*
  - $(\bar{u} + d) \models \mathcal{L}_I(\bar{s})$*;*
- $(\langle \bar{s}, \bar{g}, C, \bar{u} \rangle, a, \langle \bar{s}', \bar{g}', C', \bar{u}' \rangle) \in \rightsquigarrow$*, if* $\exists\ (\langle \bar{s}, C \rangle, \phi, a, \Omega, Y, \vartheta, \langle \bar{s}', C' \rangle) \in T$ *such that*
  - $\bar{u} \models \mathcal{L}_I(\bar{s}) \ \wedge\ \bar{u}' = \bar{u}[Y \mapsto 0] \ \wedge\ \bar{u}' \models \mathcal{L}_I(\bar{s}')$*;*
  - $\bar{g}, \bar{u} \models \phi$*;*
  - $\bar{g}' = update(\bar{g}, \Omega)$.

That is, either the system remains in the same state and time passes, or a certain transition of some STS immediately takes place.

3.4.3 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 γ, if there is a transition $(\gamma, a, \gamma') \in \rightsquigarrow$. In this case, we write $\gamma \xrightarrow{a} \gamma'$. Let $\pi = \gamma_1, a_1, \gamma_2, a_2, \ldots$ be a (possibly infinite) sequence of configurations and actions interleaved. We say that the sequence is fireable from γ_1, written as $\gamma_1 \xrightarrow{\pi}{}^*$, 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 processes and exchanged messages, but also the state of the infrastructure, i.e., queue contents. The latter information may vary with variation of the middleware implementation and therefore is not important for the verification of the business requirements of the Web service composition. Therefore, 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 part of the composition and its 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 $\pi = \gamma_0, a_0, \gamma_1, a_1, \ldots$ such that $\gamma_0 \xrightarrow{\pi}{}^*$, $\gamma_0 \in \Gamma_0$, and for each $i \geq 0$, $\sigma_i = \sigma(\gamma_i)$. We write $\sigma_0 \xrightarrow{\omega}{}^*$ to denote that the run ω is fireable from observable state σ_0.*
We call a set of runs, fireable from some initial observable state, 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.

3.5 Composition Requirements

In order to enable formal analysis of the Web service composition, it is necessary to provide a formalism for the definition of various business requirements and constraints. In particular, it is important to provide a formal language for specifying functional properties

of the composition behavior. For these purposes, we exploit the Linear-time Temporal Logic formalism. We remark, that other property modelling/specification notations may be adapted, as it is done in, e.g., [FUMK03] for Message Sequence Chart specifications.

3.5.1 Linear-time Temporal Logic

Linear-time Temporal Logic (LTL for short, [Eme90]) is the 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[2].

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 states (program counters), 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 the following sections.

More formally, the properties are defined over behaviors of the composition. The constraints on the events, their ordering, and relations are defined using combinations of temporal and boolean operators. The formula is defined inductively on a behavioral sequences $\omega = \sigma_0, a_0, \sigma_1, a_1, \ldots$ of the composition as follows:

- $(t_1 = t_2)$ – in the ground state of the first step the terms are equal;
- $action(a)$ – a is a first action;
- $state(s, i)$ – s is value of the i^{th} program counter in the first step;
- $\neg\phi$ – the property ϕ is not satisfied on the sequence;
- $\phi_1 \vee \phi_2$ – either ϕ_1 or ϕ_2 is satisfied;
- $\phi_1 \wedge \phi_2$ – both ϕ_1 and ϕ_2 are satisfied;
- $\Diamond\phi$ – property ϕ is eventually satisfied;
- $\Box\phi$ – property ϕ is satisfied at every step of the sequence;
- $\phi_1 \,\mathcal{U}\, \phi_2$ – ϕ_1 is satisfied at each step until a moment where ϕ_2 is satisfied.

[2]We remark that while in our formalism the execution may be finite terminating in some final state, the infinite execution may be emulated by adding a fake self loop to such a final state

Given this syntax, a wide range of behavioral properties and patterns may be specified, e.g., deadlock freeness, transactional consistency, data-related properties etc.

3.5.2 Property Categories

In addition to temporal dimensions, properties may belong to different categories that determine their interpretation with respect to behavior of the compositions. We distinguish between *assertion* properties and *possibility* properties.

The possibility properties describe the scenarios that are desired by the composition designers. For example, such properties may express the ability to reach a positive payment acknowledgement resulted from a business transaction, ability to perform certain operation, etc. Intuitively, the requirement is that there should exist an execution of the composition where the property holds.

Definition 3.10 (Possibility Satisfiability)
Σ_Δ *satisfies a possibility* ϕ*, written as* $\Sigma_\Delta \models_P \phi$*, iff there exists a run* $\omega \in \Xi(\Sigma_\Delta)$*, s.t.* $\omega \models \phi$*.*

On the other hand, 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 3.11 (Assertion Satisfiability)
Σ_Δ *satisfies an assertion* ϕ*, written as* $\Sigma_\Delta \models_P \phi$*, iff for all runs* $\omega \in \Xi(\Sigma_\Delta)$*,* $\omega \models \phi$*.*

Chapter 4

Formal Techniques for Composition Verification

In this chapter we present a set of techniques and approaches that are essential for the analysis of Web service compositions. These techniques are designed in order to manage certain features that are specific to the service composition problem domain. In particular, we define algorithms to manage the complexity and diversity of message exchange management in the SOA-based systems; techniques and models to deal with the arbitrary XML data and its impact on the composition behavior; and an approach to represent, verify, and compute certain time properties of the service compositions.

4.1 Analysis of Communication Models

One of the key aspects for verification of the Web service compositions is the model adopted for representing the communications among the Web services. Indeed, the actual mechanism implemented in the existing BPEL execution engines is both very complex and implementation dependent. More precisely, BPEL processes exchange messages in an asynchronous way; incoming messages go through different layers of software, and hence through multiple queues, before they are actually consumed in the BPEL activity; and overpasses are possible among the exchanged messages.

However, most of the approaches proposed for a formal verification of BPEL compositions exploit a synchronous model of communications, which does not require message queues and hence allows for a better performance in verification. This synchronous mechanism relies on some strong hypotheses on the interactions allowed in the composition: at a given moment in time, only one of the components can emit a message, and the receiver of that message is ready to accept it (see e.g., [FBS04]).

In our experience, these hypotheses are not satisfied by many Web service composition scenarios of practical relevance, where critical runs can happen among messages

emitted by different Web services. This is the case, for instance, when a Web service can receive inputs concurrently from two different sources, or when a service which is executing a time consuming task can receive a cancellation message before the task is completed. As a result, it is necessary 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.

The goal is to provide extended composition mechanisms, where the hypotheses on synchronous communications are weakened, but the communication model is kept as simple as possible. This way, an accurate modeling is possible for a wider class of service composition scenarios, while an efficient performance is still achievable in verification.

Here we propose a model of composition, which is based on a parametric definition of the communication infrastructures [KP05, KPS06]. More precisely, it is possible to define different communication models by changing the number of queues existing among the component processes and the sets of messages associated with the various queues. By increasing the number of queues, and hence by allowing more and more asynchrony in the evolution of the system, we define a hierarchy of communication models that are able to model larger and larger composition scenarios. The most restrictive model, with only one shared queue of capacity 1, is shown to be equivalent to the synchronous model of [FBS04]. 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 require that the composition models exhibit the following correctness properties. First, we consider systems with channels that grow unboundedly as "bad" systems. One of the critical problems to be addressed during the analysis is to identify and rule out these systems. Second, we require that the composition is *complete*, that is, all the messages are eventually consumed.

4.1.1 Hierarchy of Models

The definition of the composition is parametric w.r.t. a communication model. Different communication models (and hence queue structures) define different behaviors for the same composition scenario. Therefore, the result of the verification of a composite system depends on the selected communication model. In order to guarantee the correctness of the verification, we have to make it sure that the selected communication model allows for all the behaviors that are compatible with the assumptions presented above.

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 communication model 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.

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 4.1 *We say that a configuration* γ_2 *of* Σ_{Δ_2} simulates *a configuration* γ_1 *of* Σ_{Δ_1}, *written as* $\gamma_1 \preceq \gamma_2$, *iff*

- $\sigma(\gamma_1) = \sigma(\gamma_2)$,
- *for any* a, *for any* γ_1', *if* $\gamma_1 \xrightarrow{a} \gamma_1'$, *then there exists* γ_2', *such that* $\gamma_2 \xrightarrow{a} \gamma_2'$, *and* $\gamma_1' \preceq \gamma_2'$.

We say that Σ_{Δ_2} simulates Σ_{Δ_1}, *written as* $\Sigma_{\Delta_1} \preceq \Sigma_{\Delta_2}$, *iff for any* $\gamma_{01} \in \Gamma_{01}$ *there exists* $\gamma_{02} \in \Gamma_{02}$ *such that* $\gamma_{01} \preceq \gamma_{02}$.

We say that Σ_{Δ_1} *and* Σ_{Δ_2} *are* bisimilar, *written as* $\Sigma_{\Delta_1} \approx \Sigma_{\Delta_2}$, *iff* $\Sigma_{\Delta_1} \preceq \Sigma_{\Delta_2} \wedge \Sigma_{\Delta_2} \preceq \Sigma_{\Delta_1}$.

Proposition 4.1
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* $\Xi(\Sigma_{\Delta_1}) = \Xi(\Sigma_{\Delta_2})$.

When the simulation relation among two communication models Δ_1 and Δ_2 holds for any set of STSs, we say that the model Δ_2 is *more general* than the model Δ_1.

Definition 4.2 *Communication model* Δ_2 simulates *model* Δ_1, *written as* $\Delta_1 \sqsubseteq \Delta_2$, *if for any composition 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 will show that there is a "most general" model, that is the model Δ_{MG}, such that for any other model Δ holds $\Delta \sqsubseteq \Delta_{MG}$.

Definition 4.3 *The* Most General Communication Model *(MG-model) is a communication model $\Delta_{MG} = \langle B, \mathcal{L}_M, \mathcal{L}_O \rangle$, with 1 unordered queue, $b = \infty$, and $\mathcal{L}_M(\alpha_i) = 1$.*

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 4.2 *For any communication model Δ, $\Delta \sqsubseteq \Delta_{MG}$.*

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

Definition 4.4 *A communication model Δ is said to be* adequate *for the given composition scenario if $\Sigma_\Delta \approx \Sigma_{\Delta_{MG}}$.*

Model Δ_{MG} defines the most liberal policy for the message processing: each message stored 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 as the one allowed in the model.

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

Theorem 4.1 *Consider two communication models $\Delta_1 = \langle B_1, \mathcal{L}_{1M}, \mathcal{L}_{1O} \rangle$ and $\Delta_2 = \langle B_2, \mathcal{L}_{2M}, \mathcal{L}_{2O} \rangle$. If for each queue q_{2i} 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 $\Delta_1 \sqsubseteq \Delta_2$.

The theorem may be easily understood on the following example. Consider a model Δ_1 with one ordered queue q_1 with alphabet $M = \{\alpha_1, \alpha_2\}$, and a model Δ_2 with two queues q_{21} and q_{22} with alphabets $\{\alpha_1\}$ and $\{\alpha_2\}$ respectively. Indeed, if an input action is allowed in the composition under model Δ_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 $\overrightarrow{\mu_1}$ 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 α_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.

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 4.5 *The* synchronizable communication model *is* $\Delta_1^1 = \langle B, \mathcal{L}_M, \mathcal{L}_O \rangle$*, with* $B = \langle 1 \rangle$ *and* $\mathcal{L}_M(\alpha) = 1$ *for all messages* α*.*

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 Δ_1^1 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. When the verification properties are defined on the set of conversations (i.e., only sequences of message exchanges), and the composition appears to be complete under Δ_1^1 model, then one can use the synchronous product for the composition analysis.

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. As a consequence, the compositions, for which this model was proved to be adequate, are very robust and exhibit the same behavior on all the implementations of BPEL engines. 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. [FBS04]). Each participant is equipped with separate queue storing messages from all the partners.

Definition 4.6 *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.* $\overleftarrow{\mu_\alpha} \in \mathcal{I}^i$*.* $\mathcal{L}_M(\alpha) = q_i$*.*

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. Similar considerations also hold for the next model we present.

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 [BZ83], 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 4.7 *A* mutually ordered asynchronous communication model *of* n *STSs is* $\Delta_{mo} = \langle B, \mathcal{L}_M, \mathcal{L}_O \rangle$*, with* $n^2 - n$ *ordered queues denoted as* $q_{i,j}$ *(*$i \neq j$*), s.t.* $b_{i,j} = \infty$*, and* $\forall\ \alpha$*.* $\overleftarrow{\mu_\alpha} \in \mathcal{I}^j \wedge \overrightarrow{\mu_\alpha} \in \mathcal{O}^j$ *iff* $\mathcal{L}_M(\alpha) = q_{i,j}$*.*

We conclude this section with the overall hierarchy of the models defined:

$$\Delta_1^1 \sqsubseteq \Delta_{lo} \sqsubseteq \Delta_{mo} \sqsubseteq \Delta_{MG}\ .$$

We remark that the GTS formalism allows for potentially infinite number of models to be defined. The MG-model is the upper bound of this construction and we use this fact in the adequacy analysis presented below.

4.1.2 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 $\Delta_1, \Delta_2, \ldots, \Delta_{MG}$ such that $\Delta_i \sqsubseteq \Delta_{i+1}$;
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.

The adequacy 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. For the sake of simplicity we omit here the definition and description of the algorithm. See [KPS06] for the details.

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. Indeed, the graph is finite, and actual queue bounds may be extracted by analyzing reachable states thus allowing for finite representation of the composition model in the model checker specifications.

As we mentioned above, the results of the verification may be affected by the data flow specified in the BPEL code. For this reason the composition obtained after steps specified above is equipped with data-related constructs, and the resulting model is analyzed using model checking techniques. Below we give some results that relates the models with and without data (the composition of skeletons).

The first step is to perform the adequacy analysis of the skeleton models, i.e., the STS without variables and data conditions. Under certain conditions, boundedness and adequacy of some model of the skeleton, implies also boundedness and adequacy in the full composition, thus justifying the skeleton-based approach. Note, however, that the opposite proposition does not hold.

Theorem 4.2 *If the skeleton of the composition under the MG-model is bounded, then the (complete) composition under the MG-model is also bounded.*

The adequacy of the skeleton of the communication model in the skeleton of the composition does not directly implies the adequacy of the model in the composition itself. More precisely, the following theorem defines the required analysis extension. We denote the number of messages of type μ stored in a queue q as $|q|_\mu$.

Theorem 4.3 *If the skeleton of the communication model is adequate for the skeleton of the composition, and each reachable state of does not contain more than one message of*

the same type in the same ordered queue, then the communication model is also adequate for the (complete) composition.

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 Δ is ordered or not, which can be done statically.

4.2 Data Flow Analysis

Along with the definition of the control flow, Web service compositions define also a data flow among the component services. While the former is used to specify the order of the activities carried out by the participants, the latter defines the exchanged data values. Clearly, the data flow is as important for the static analysis of Web service compositions as the control flow, but most of the existing approaches ignore data, thus potentially invalidating the analysis results.

The necessity to manage data, however, brings the problem that 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 manipulating operations is complex (e.g., the XPath expressions used in BPEL specifications). This restricts the applicability of traditional formal analysis techniques that rely on simple and finite system representations. Abstraction techniques such as the ones defined in [GS97, CCG+04] are needed to take into account only those data-related properties that are relevant for the verification, and discard the aspects that are irrelevant.

The abstraction-based techniques are widely used in the traditional domains, like program verification. Differently from these domains, Web service-based systems exhibit a specific feature that should be taken into account. The coarse granularity and distributed nature of the Web service (composition) specifications implies that the internal details of the implementations of the component services may be hidden to the analysts. This lack of knowledge is useful as far as it allows for a higher-level definition of composite business processes which is independent from the implementation details of the components. However, it may also make it impossible to prove the correctness of the composition. In this case, the system model should be refined by introducing those assumptions on the internal of service implementations that are necessary to guarantee a correct composition.

We see the analysis of Web service compositions in presence of data as an iterative process, where the verification of expected properties is interleaved with the elicitation of the assumptions on the data flow 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. We remark that 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.

Here we present a theoretical framework for the abstraction-based verification of Web service compositions that supports the iterative analysis process [KP06c]. In this framework we address the following goals:

- we provide a foundation for the modeling and analysis of data in compositions through abstractions;
- we provide the ability to specify and verify different behavioral properties in this setting;
- we manage the incompleteness in the composition specification allowing the definition of assumptions on the hidden properties of the systems;
- we support an iterative refinement process by combining the abstraction-based verification with the requirements extraction.

4.2.1 Data Abstraction Model

The verification of the STS composition is not doable in general due to the fact that the data types are infinite and the functions are potentially 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).

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 3.3.1.

For the abstraction 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 considered propositions as $\mathcal{P}^A$.

The set of propositions used in the specification 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. Without loss of generality, we assume that such an interpretation exists for the verification properties, while for the transition conditions we give an abstraction-specific interpretations in Section 4.2.1.

Definition 4.8 (Abstract Formula) *Given a LTL formula ϕ, its* abstract interpretation *with respect to the set of propositions $\mathcal{P}^A$ is an LTL formula $[\phi]$ obtained as follows:*

1. *pushing negations inside up to a propositional subformula;*
2. *each propositional subformula ψ is replaced by $\psi' = \bigvee_i \bigwedge_j p_{ij}$, with $p_{ij} \in \mathcal{P}^A$, s.t. $\psi' \Leftrightarrow \psi$.*

Proposition 4.3 *$\Sigma \models [\phi]$ if and only if $\Sigma \models \phi$.*

Given a set of propositions $\mathcal{P}^A$, a *valuation* of the propositions 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 $consistent(V)$, if and only if $\mathcal{I}(V) \neq \emptyset$. We write $V \models \phi$ when $V(\phi) = true$.

Abstract Composition

Analogously to the concrete composition model, the abstract composition model defines the evolution 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 should be finite.

Abstract configuration in this model is a tuple $\langle \bar{s}, V, C, \bar{u} \rangle$, where V is the valuation of propositions. We say that a transition may be performed given a valuation of abstract propositions V, if its condition on data is *applicable* with respect to this valuation, written as $applicable[\phi_D](V)$. The *effect* of the transition defines the modification of the valuation according to the transition assignment, denoted as $exec[\Omega](V)$.

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 4.9 (Abstract Global Transition System)
The semantics of an abstract CSTS $\langle GS, GS_0, \mathcal{V}, \mathcal{X}, \mathcal{A}, T, \mathcal{L}_I \rangle$ is defined as an abstract

global transition system $\Sigma^A_\Delta = (\Gamma^A, \Gamma^A_0, \rightsquigarrow^A)$, *where* Γ^A *is a set of configurations of the form* $\langle \bar{s}, V, C, \bar{u} \rangle$, $\Gamma^A_0 \subseteq \Gamma^A$ *is a set of initial configurations with* $\bar{u} = \langle u_{10}, \ldots, u_{n0} \rangle$, *and* $\rightsquigarrow^A \subseteq \Gamma^A \times \{\mathcal{A} \cup \text{TICK}\} \times \Gamma^A$ *is a transition relation such that:*

- $\langle \bar{s}, V, C, \bar{u} \rangle \xrightarrow{\text{TICK}} \langle \bar{s}, V, C, \bar{u} + d \rangle$, *if*
  - $\forall\ (\langle \bar{s}, C \rangle, \phi_T \wedge \phi_D, a, \Omega, Y, \vartheta, \langle \bar{s}', C' \rangle) \in T$, *if* $\vartheta = true$ *then* $\bar{u} \not\models \phi_T$ *or* $\neg applicable[\phi_D](V)$;
  - $(\bar{u} + d) \models \mathcal{L}_I(\bar{s})$;
- $\langle \bar{s}, V, C, \bar{u} \rangle \xrightarrow{a} \langle \bar{s}', V', C', \bar{u}' \rangle$, *if* $\exists\ (\langle \bar{s}, C \rangle, \phi_T \wedge \phi_D, a, \Omega, Y, \vartheta, \langle \bar{s}', C' \rangle) \in T$ *such that*
  - $\bar{u} \models \mathcal{L}_I(\bar{s})\ \wedge\ \bar{u}' = \bar{u}[Y \mapsto 0]\ \wedge\ \bar{u}' \models \mathcal{L}_I(\bar{s}')$;
  - $\bar{u} \models \phi_T\ \wedge\ applicable[\phi_D](V)$;
  - $V' = exec[\Omega](V)$.

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 state is changed. A behavior of the abstract composition Σ^A is defined in the usual way.

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. For this purposes we try to build an abstraction in such a way that it 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 *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 4.10 (K-Interpretation) *Given a proposition valuation* V, *its* K-interpretation, *denoted as* $\mathcal{I}^K(V)$, *is the set of global ground states* $\bar{g} \in \mathcal{I}^K(V)$ *s.t.* $\forall\ p \in \mathcal{P}^A$, *if* $V(p) = true$ *then* $\Gamma_{\bar{g}}(p) = true$.

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 4.11 (Applicability and execution in K-Model)
A data condition ϕ_D is applicable *in V, written as $applicable[\phi_D](V)^K$, iff $\forall\ \bar{g} \in \mathcal{I}(V)$, $\Gamma_{\bar{g}}(\phi_D) = true$.*

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

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

Under the applicability condition defined as above, a situation is possible where the execution of the abstract model reaches a state with branching, and no transition is allowed since neither the condition or 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 state.

Let us define a verification problem of K-model of STS composition. We say that the K-model of STS composition satisfies the possibility ϕ 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 ϕ if there is no run that satisfies $[\neg\phi]$.

Definition 4.12 (Satisfiability in K-Model)
The possibility ϕ is satisfied in K-model, written as $\Sigma^K \models_P \phi$, iff there exists a run ω^K of Σ^K, s.t. $\omega^K \models [\phi]$.
The assertion ϕ is satisfied in K-model, written as $\Sigma^K \models_A \phi$, iff for each ω^K of Σ^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 paper.

Theorem 4.4
Given an assertion ϕ, if $\Sigma^K \not\models_A \phi$, then $\Sigma \not\models_A \phi$.
Given a possibility ϕ, if $\Sigma^K \models_P \phi$, then $\Sigma \models_P \phi$.

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 $\mathcal{I}(V)$.

Definition 4.13 (B-Interpretation) *Given a proposition valuation V, its* B-interpretation, *denoted as $\mathcal{I}^B(V)$, 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 "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 4.14 (Applicability and execution in B-Model)
A data condition ϕ_D is applicable *in V, written as $applicable[\phi_D](V)^B$, iff $\exists\ \bar{g} \in \mathcal{I}(V)$ s.t. $\Gamma_{\bar{g}}(\phi_D) = true$.*

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

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.

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 4.15 (Satisfiability in B-Model)
The possibility ϕ is satisfied in B-model, written as $\Sigma^B \models_P \phi$, iff there exists a run ω^B of Σ^B, s.t. $\omega^B \models [\phi]$.
The assertion ϕ is satisfied in B-model, written as $\Sigma^B \models_A \phi$, iff for each ω^B of Σ^B, $\omega^B \models [\phi]$.

Theorem 4.5
Given an assertion ϕ, if $\Sigma^B \models_A \phi$, then $\Sigma \models_A \phi$,
Given a possibility ϕ, if $\Sigma^B \not\models_P \phi$, then $\Sigma \not\models_P \phi$.

Relations between Abstraction Models

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

Theorem 4.6 *Given an assertion ϕ,*

$$\Sigma^B \models_A \phi \;\Rightarrow\; \Sigma \models_A \phi \;\Rightarrow\; \Sigma^K \models_A \phi.$$

Given a possibility ϕ,

$$\Sigma^K \models_P \phi \;\Rightarrow\; \Sigma \models_P \phi \;\Rightarrow\; \Sigma^B \models_P \phi.$$

The theorem suggests that the interplay of the two models of abstractions is used in order to put bounds on the satisfiability of the property. 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, he 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.

Construction of the Abstraction

In this section we present the translation procedure that allows 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 above);
- 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.

4.2.2 Representation of Data Assumptions

Due to intrinsic information incompleteness of the Web service specification, it is often hard 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. Second, the assumptions may describe the values that the output of the service function *may* have. This kind of assumptions allow for modelling non-deterministic data operations that, nevertheless, take place in real executions (e.g., for modelling "opaque" assignments in BPEL). 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 $\mathcal{P}_{Inv}$ and the properties of the second class as $\mathcal{P}_{Alt}$.

Definition 4.16 (Implementation Assumptions) *Let $f(x_1, \ldots, x_n)$ be a function. An* invariant assumption *in $\mathcal{P}_{Inv}$ over the function has the form*

$$\textit{precond-expr} \rightarrow \textit{effect-expr},$$

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 $\mathcal{P}_{Alt}$ over the function has the form*

$$\textit{precond-expr} \rightarrow \textit{alternative}\ (\sqcup\ \textit{alternative})^*,$$

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.

The constraints modelled in this way allows for refinement of the composition specification, representing those properties of the composition that are crucial for the correct implementation. They may be further exploited for the run-time analysis of the composition using monitoring techniques.

4.3 Analysis of Time Properties

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 this setting. Indeed, in many scenarios we expect that a Web service composition satisfies some global timed constraints, and these constraints can be satisfied only if all the services participating to the composition are committed to respect their own local timed constraints. Consider for instance an e-government scenario, where the distributed business process requires the composition of information systems and functionalities provided by different departments or organizations (here, we will consider one of such scenarios, consisting in providing the authorization to open a site for the disposal of dangerous waste). The composite service can comply with the timed commitments with respect to the national regulations (e.g., the duration of document analysis phase) 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. In e-government scenario, for instance, 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. However, the computation of these durations by trial and error search of the corresponding values is inherently incomplete and highly inefficient.

In this section we present techniques [KPP06b, KPP06a] for modeling, validating and computing the time-related properties of Web service compositions defined by a set of BPEL processes. 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., they refer to the time required by the participating actor to carry out their tasks and take their decisions, and to the assumptions and constraints on these times that guarantee a successful execution of the distributed business processes. In e-government scenarios, there times are measured in hours and in days. The "technical" times, which are required, for instance, by the communications among BPEL processes and by the BPEL engines to manage incoming and outgoing message, are orders of magnitude smaller (seconds if not milliseconds) and can be neglected in these scenarios.

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

quantitative characteristics.

BPEL Duration Annotations

BPEL duration annotations allow for the representation of 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. Such constraints are conjunctions of the clauses of the form $dur \sim c$, where $\sim \in \{<, >, \leq, \geq, =\}$.

When applied to an atomic activity, this annotation is semantically equal to the sequence of two transitions. The first one is an instant transition and resets the clock x_t. The second transition has the guard that evaluates to true, if the value of the clock x_t satisfies the duration constraints. An corresponding invariant is defined in the intermediate state.

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. 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 ($start\_a$) to its end (end_a) has the required duration. A corresponding automaton is generated for the constraint, and the synchronous (i.e., lock step) product of the GTS Σ_Δ and the automaton is built. This constraint is specified in the duration calculus, which we will describe in the next section.

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 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) [CCR91]. 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 following. 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 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:

$$d := \lceil p \rceil^0 \mid \lceil\lceil p \rceil\rceil \mid d_1 \frown d_2 \mid d_1 \ \wedge \ d_2 \mid \neg d \mid len \ \sim \ c \mid len(p) \sim \ c$$

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

Intuitively, $\lceil p \rceil^0$ means that the interval is a point and at that point the proposition p holds. $\lceil\lceil p \rceil\rceil$ requires the propositions to hold at each position of the interval, while $d_1 \frown d_2$ 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.

Consider, for example, the requirement stating that the interval from the state a to c consists of two subintervals with a state b in between, such that the first should not exceed 30 days, and the second should last at least 10 days. This requirement may be expressed with the following DC formula:

$$\begin{array}{l} \Box(\lceil state(a) \rceil^0 \frown true \frown \lceil state(c) \rceil^0 \rightarrow \\ \quad (len \ \leq 30) \frown \lceil state(b) \rceil^0 \frown (len \ \geq 10)) \end{array}$$

4.3.2 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.

Discrete Time Representation

In the implementation we adopt discrete model of time, and use a corresponding subset of *Quantified Discrete-time Duration Calculus* (QDDC, [Pan01]) to express complex time requirements under this model (the analysis based on the dense model of time under certain conditions may be implemented in a similar way. See [Pan02] for details).

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 [AD94] 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: $\langle \bar{s}, \bar{g}, C, \bar{u} \rangle \stackrel{\text{TICK}}{\rightarrow} \langle \bar{s}, \bar{g}, C, \bar{u} + 1 \rangle$.
- 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.

min_duration$(start, final)$
  $R := start \cap reachable(final)$
  $min := 0$
  if $(R = \emptyset)$ **return** ∞
  loop
    $R := immediate(R)$
    if $(R \cap final \neq \emptyset)$ **return** *min*
    $R := delayed(R)$
    $min := min + 1$
  end loop

max_duration$(start, final)$
  $R := start \cap reachable(final) \cap \neg final$
  $max := 0$
  $R' := \emptyset$
  while$(R \neq \emptyset \wedge R \not\subseteq R')$ **do**
    $R := immediate(R) \cap \neg final$
    $R' := R' \cup R$
    $R := delayed(R) \cap \neg final$
    $max := max + 1$
  end while
  if $(R = \emptyset)$ **return** max
  else return ∞

Figure 4.1: Min/Max duration algorithms

When the composition is finitely represented with a set of atomic propositions (e.g., using the abstraction techniques), the requirements expressed as QDDC formulas may be effectively expressed and analyzed [Pan01].

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. This algorithm relies on previous results of [Pan04], where the computation of extremal values for *synchronous* systems is addressed, and on the symbolic condition count algorithms of [CCMH94].

Intuitively, the algorithm performs as follows. First, the initial system M and a property p are transformed into a system $M'(start, final)$. Minimal (maximal) duration of interval between a state satisfying formula $start$ and a state satisfying formula $final$ in M' is equivalent to the minimal (maximal) duration of interval satisfying p in M. We refer the reader to [Pan04] for the details of this transformation.

Second, the actual extremal value is computed in M' using the minimal/maximal duration algorithms that measure time intervals between states satisfying $start$ and $final$ (Fig. 4.1).

The minimal duration algorithm performs as follows. Initially, the set of states R is a set that satisfies $start$ and from which the states that satisfy $final$ is reachable ($reachable(final)$). 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 duration 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 states R' that contains all the states visited so far. If the cycle is detected ($R \in R'$), the algorithm returns *infinity*.

Chapter 5

Formal Analysis of Web Service Compositions

In this chapter we present the various kinds analysis enabled by the presented framework. First, the framework allows for the verification of the Web service composition against a set of business requirements and constraints. Second, it provides a way to perform choreography analysis, that is to check the correctness of the choreography model (realizability analysis), and the correspondence of a composition implementation to the choreography (conformance checking). Third, using the LOTOS models of the participating services, it is possible to apply the forms of analysis based on the notions of process simulation and bisimulation.

5.1 Verification of Composition Requirements

In order to perform the verification of composition requirements, the composition model is first analyzed using the techniques defined above. In particular, an adequate communication model is defined and an abstract representation of data is constructed.

The verification procedure uses the following components as input:

- composition specification Σ_{Δ} that represents a composition of BPEL processes, the communication model applied, set of abstract propositions, etc;
- set of various constraints $\mathcal{P}^C$ and assumptions, expressed as LTL formulas, data constraints, timed DC formulas;
- set of target properties (assertions $\mathcal{P}^A$ and possibilities $\mathcal{P}^P$) to be verified against the composition (LTL or DC);
- set of timed DC properties $\mathcal{P}^V$ to be measured using the quantitative algorithms.

Given these inputs the verification is performed as follows.

Constraining the specification. The set of constraints $\mathcal{P}^C$ express the assumed behavior of the system. That is, in the analysis we should consider only those behaviors that satisfy such a set of properties. In order to represent this restriction, each constraint property $p_i \in \mathcal{P}^C$ is translated into the finite state automaton $\mathsf{A}(p_i)$ that recognizes all and only the behaviors that satisfy p_i. Finally, a synchronous (i.e. lock-step) product $(\Sigma_\Delta \times (\mathsf{A}(p_1) \times \cdots \times \mathsf{A}(p_n)))$ of the specification and the properties automata is constructed. Indeed, this product describes the specification that satisfies all this constraints. This product is used for further analysis of the composition. If the product is empty, the specification is inconsistent, and the constraints should be relaxed.

Verification of assertion properties. The assertion properties $\mathcal{P}^A$ represent requirements that are expected to hold on every behavior of the system. In order to verify such a property $p \in \mathcal{P}^A$, an automaton corresponding to the negation of the property is created and a synchronous product $(\Sigma_\Delta \times \mathsf{A}(\neg p))$ of the specification and the automaton is built. If this product is not empty, then the behavior of the composition violates the property p. A path in the product represent a counterexample of the property violation, that is, an execution if the system where the property does not hold.

Verification of possibility properties. The possibility properties $\mathcal{P}^P$ on the contrary represent requirements that are expected to hold on some behavior of the system. In order to verify such a property $p \in \mathcal{P}^P$, an automaton corresponding to the property is created and a synchronous product $(\Sigma_\Delta \times \mathsf{A}(p))$ of the specification and the automaton is built. If this product is not empty, then there exists an execution of the composition that satisfies the possibility.

Computation of timed properties. The quantitative analysis represented in the previous chapter may be used to *measure* a certain property $p_i \in \mathcal{P}^V$. Following the above procedure, the (possibly constrained) composition model is transformed into a new model and the minimum/maximum duration algorithms are applied. The time bounds provided represent the corresponding interval where the property holds.

Extraction of data constraints. Due to a possible incompleteness of the composition specification, the verification of a certain may fail to provide a sound answer. Indeed, as we discussed before, it is possible that the assertion property is violated by B-model, but is not violated by the K-model of abstraction. In this case further refinement is needed, and additional data constraints should be introduced. This procedure is repeated until the result of the analysis is precisely defined. The outcome of this iterative procedure is not only, a correct composition, but a set of data constraints that ensure this correctness. These

data constraints should be further monitored when the composition is being executed in real settings.

5.2 Choreography Analysis

Choreography analysis aims at checking a certain correspondence between the choreography model (that is, a blueprint, global model of the composition) and its possible implementation (that is, a set of real services or service models). This includes conformance checking [GP03], replaceability analysis [BCPV04], and realizability analysis [FBS04]. The ability to perform these kinds of verification relies on the existence of a certain formal model of the choreography specification.

5.2.1 Formal Model of Choreography Specification

Choreography model describes the Web service composition from a global point of view. It describes global vision of the conversations between the participants, and the evolution of the distributed application. In this view, the primary activity is an interaction between two partners, which is seen as an atomic action.

We model message communications actions as *interactions* defined on a set of service operations (or message types) M. The signature of the interaction has the form $(r_s, r_d, \mu, \bar{v}_s, \bar{v}_d)$, where r_s and r_d are the roles of the sender and receiver respectively, μ is the service operation, and variables $\bar{v}_d$ of the receiver are populated with the values of the corresponding variables $\bar{v}_s$ of the sender. Set of interactions is denoted as $\mathcal{A}_O$.

We also define *internal actions* $\mathcal{A}_\tau$, which are used to represent evolutions of the system that do note involve interactions between services. An internal action τ has the form (R_τ, τ), where $R_\tau \subseteq R$ denotes a subset of roles that perform an action, and τ is used to denote the internal action itself. The set of all actions is denoted as $\mathcal{A}$.

We model a choreography behavior as a *Choreography Transition System* (ChorTS).

The model of transitions and the semantics is defined analogously to those of the Web service composition model defined previously. Informally, we represent a *global state* of the choreography as a vector $\bar{s} = \langle s_1, \ldots, s_n \rangle$, where s_i is a local state of the role r_i. We denote a vector with component s_i updated to s_i' as $\bar{s}[s_i/s_i']$. The behavior of the choreography is defined by the *choreography transition relation* T^C. The relation defines *conditions*, under which the action can be performed, and *effects* of these executions, which specify the modification of the states and variables of the participants.

Definition 5.1 (ChorTS) *A* global transition system *representing the choreography of* n *roles is a tuple* $Chor = \langle \bar{\mathcal{S}}, \bar{\mathcal{S}}_0, \mathcal{V}, \mathcal{X}, \mathcal{A}, T^C, \mathcal{L}_I \rangle$*, where*

- $\mathcal{R} \subseteq \bar{\mathcal{S}} \times \Phi \times \mathcal{A} \times \Omega^* \times \{true, false\} \times \bar{\mathcal{S}}$ *is a global transition relation. A transition* $t = (\bar{s}, \phi, a, \Omega, Y, \vartheta, \bar{s}')$ *is in* T^S*, iff*
  - (R_τ, τ)*, and* $\bar{s}' = \bar{s}[s_i/s_i']$ *for each* $r_i \in R_\tau$
  - $a = (r_a, r_b, \mu, \bar{v}_s, \bar{v}_d)$*,* $\bar{s}' = \bar{s}[s_a/s_a', s_b/s_b']$.

The data and time semantics of the transition system is defined as previously. We remark, that the representation of the choreography does not require queues. Indeed, the interactions are represented atomically and therefore the messages are consumed immediately after the emission.

5.2.2 Realizability of Choreography Models

An important issue emerging from the choreography modelling, is the protocol realizability, i.e., the possibility to extract the local specifications of the participants, so that their interactions preserve certain crucial properties of the global description.

This problem is made difficult by several crucial factors. First, the behavior of the application strongly depends on the way the services are exchanging the information, that is the *communication model* of the composition. Second, the strictness to which the application should satisfy the ordering constraints on messages and/or internal activities may differ from one scenario to another. For instance, in certain cases we may require the order of all activities is respected completely, while in another cases allow for reordering of interaction between independent partners. In order to address this diversity, we present a hierarchy of realizability notions that allows for capturing a variety of the choreography properties, thus providing a basis for a more flexible analysis [KP06a]. For the determining a corresponding level of realizability we use the hierarchy of communication models, which allows express the realizability problem in terms of differences between communication models.

Realizability Hierarchy

The realizability problem it consists in deciding whether there is a way to extract the local implementations of the participating roles such that, when composed together, they satisfy the protocol specifications. We refer to these local implementations as *projections*. Sets of such projections will be used as the implementation candidates for the composition.

Intuitively, a behavior of a projection is constructed from those transitions of the global protocol, in which the participant is involved. The internal action is projected onto an internal action of each role. The interaction action is projected onto the input action of the receiver, and on the output action of the sender. Given the set of all role projections of a global protocol, we can compose them into a CSTS. In the following we denote the composition of role projections under the model Δ as Σ^p_Δ.

Given a choreography model, we define the following levels of realizability (see [KP06a] for more details):

- *synchronous realizability*. This notion requires that the composition of the projections behaves exactly as the given global specification regardless the communication model that is applied. It does not allow any reorderings of actions in the composition of projections.

- *strong realizability*. The restrictions imposed by the synchronous realizability are often too strong for the implemented system. Indeed, it requires that the order of internal and external actions is respected by the implementation, or that the next emission cannot start before the previous message was received, even if the acting participants are independent. The notion of strong realizability relaxes these constraints. The ordering restrictions concern only the communication actions, and not the internal ones. Intuitively, a protocol is strongly realizable if the set of conversations of the protocol and of the compositions is the same, the composition is bounded, and all the emitted messages are received.

- *Local realizability*. Strong realizability may appear to be too restrictive for a wide class of choreography scenarios. Indeed, it does not allow for concurrent emissions and receptions of messages by independent processes. For instance, if role A interacts with B, and then C interacts with D, then the global order cannot be preserved by the composition of projections. In many cases, however, this may be irrelevant. Moreover, since the variables are local for each role, the behavior and the information of the local participant is not affected by these reorderings. We relax the notion of strong realizability by omitting the requirement of conversation equivalence. However, the local behavior of each role should not be affected by possible reorderings of message emissions.

- *weak realizability*. The least restrictive model of realizability further relaxes the ordering constraints, requiring only that the message ordering of the interactions among a pair of the participants is preserved. That is, each participant sending messages to its partner knows that they will be processed and managed in turn.

Realizability Checking

In order to determine the corresponding level of realizability we exploit the hierarchy of communication models. Intuitively, for each of the above notions a corresponding communication model is assigned, that exhibits the same properties with respect to various types of message and activity reorderings. If the communication model is adequate for the composition of projections, then the reorderings do not appear and the corresponding level of realizability is detected. More precisely, the following communication models are used:

- *Synchronizable communications model* is used for synchronous realizability. This is a model $\Delta_1^1 = \langle B, \mathcal{L}_M, \mathcal{L}_O \rangle$, with $B = \langle 1 \rangle$ and $\mathcal{L}_M(\mu) = 1$ for all operations μ. If the protocol $ChorTS$ is synchronously realizable, then the model Δ_1^1 is adequate for the composition of the local projections of $ChorTS$.

- *Globally ordered communication model* is used for strong realizability. This is the model $\Delta_{go} = \langle B, \mathcal{L}_M, \mathcal{L}_O \rangle$, with one ordered queue, such that $B = \langle \infty \rangle$ and $\mathcal{L}_M(\mu) = 1$ for all operations μ. The protocol is strongly realizable iff Δ_{go} is adequate for the composition of projections.

- *Locally ordered communication model* is used for local realizability. This 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. $\overleftarrow{\alpha} \in \mathcal{I}^i$, $\mathcal{L}_M(\alpha) = q_i$. The protocol is locally realizable iff Δ_{lo} is adequate for the composition of projections.

- *Mutually ordered communication model* is used for local realizability. This is a model $\Delta_{mo} = \langle B, \mathcal{L}_M, \mathcal{L}_O \rangle$, with $n^2 - n$ ordered queues denoted as $q_{i,j}$ $(i \neq j)$ s.t. $b_{i,j} = \infty$, and $\forall\ \alpha$, $\overleftarrow{\alpha} \in \mathcal{I}^j \wedge \overrightarrow{\alpha} \in \mathcal{O}^j$ iff $\mathcal{L}_M(\alpha) = q_{i,j}$. The protocol is weakly realizable iff Δ_{lo} is adequate for the composition of projections.

Using the algorithms of [KPS06, KP06a] the realizability is effectively analyzed by checking the adequacy of the corresponding communication model.

5.2.3 Conformance Checking

Conformance Checking refers to the verification that the joint behavior of the composition of the service implementations corresponds to that described in the choreography. However, it does not amount only to check the correspondence between the sequences of externally observable message exchanges generated by the composition of service implementations and the collaboration protocol specification. It is also necessary to verify that the information of the protocol is being managed and distributed accordingly, and that the participants have a common view of the business data described in the choreography. The management of business information in conformance testing is complicated by the fact that WS-CDL [W3C05] allows for specifying in a declarative way that certain pieces of information should be synchronized either as a result of a certain data exchange (interaction alignment) or of the protocol execution as a whole (choreography coordination), without explicitly describing and constraining the mechanisms that should implement them.

In our work [KP06b] we extended our analysis framework to allow for the verification of the conformance between the collaboration specification and the composition of service implementations. The key feature of this analysis is the ability to model and analyse asynchronous interactions. In this extension, we address the conformance checking problem as follows. We define the choreography conformance as a kind of bisimulation re-

lation, emphasizing the asynchrony of the message communications. We also present formal definitions for the most common information alignment requirements, such as the interaction coordination alignment rules presented by WS-CDL. Furthermore, we define a symbolic representation of the underlying models, and propose finite-state model checking techniques for verifying the conformance between the implementing composition and the choreography specifications.

Conformance Relation

We extends the notion of conformance between choreography and orchestration (i.e. implementation specification), introduced in [BGG+05] as a bisimulation-like relation, with the modelling of asynchronous interaction, and data related constructs. We require that the resulting composition is *complete*, *bounded*, and the (relevant part of) observable behavior of the implementation is *similar* to the behavior of the choreography specification.

More formally, we define the notion of conformance as follows. In order to hide irrelevant operations of the implementation, we use the operator $[\cdot]$. That is, given some action a, we write $[a] = \mu$, if $a = \overrightarrow{\mu}$ and $\mu \in M^C$ (operations mentioned in the choreography), and $[a] = \tau$ otherwise. The conformance relation requires that conversations of the implementing composition reflects all and only the conversations of the choreography.

Definition 5.2 (Conformance Relation) *Let γ^C and γ^I be configurations of implementing composition and choreography respectively. We say that the relation $R(\gamma^C, \gamma^I)$ is a* conformance relation *if for any transition label a*

- *if* $\gamma^C \xrightarrow{a} \gamma_1^C \ \wedge \ a = \mu$, *then* $\gamma^I \xrightarrow{\tau}{}^* \gamma_2^I \ \wedge \ \gamma_2^I \xrightarrow{\overrightarrow{\mu}} \gamma_1^I \ \wedge \ R(\gamma_1^C, \gamma_1^I)$;
- *if* $\gamma^I \xrightarrow{a} \gamma_1^I \ \wedge \ [a] = \mu$, *then* $\gamma^C \xrightarrow{\tau}{}^* \gamma_2^C \ \wedge \ \gamma_2^C \xrightarrow{\mu} \gamma_1^C \ \wedge \ R(\gamma_1^C, \gamma_1^I)$;
- *if* $\gamma^C \xrightarrow{a} \gamma_1^C \ \wedge \ a = \tau$, *then* $\gamma^I \xrightarrow{\tau}{}^* \gamma_1^I \ \wedge \ R(\gamma_1^C, \gamma_1^I)$;
- *if* $\gamma^I \xrightarrow{a} \gamma_1^I \ \wedge \ [a] = \tau$, *then* $\gamma^C \xrightarrow{\tau}{}^* \gamma_1^C \ \wedge \ R(\gamma_1^C, \gamma_1^I)$.

We write $\Sigma^C \approx \Sigma^I$ if there exists a conformance relation R, such that any initial configuration of Σ^C conforms to some initial configuration of Σ^I, and vice versa.

Definition 5.3 (Asynchronous Choreography Conformance) *An implementing composition Σ^I is* asynchronously conformant *to the choreography Σ^C, if Σ^I is complete, is bounded, and $\Sigma^C \approx \Sigma^I$.*

The verification of the asynchronous conformance relation between the implementation and choreography models may be done symbolically, based on the abstractions for these models. The symbolic algorithm, adopted for the conformance checking analysis is presented in [BCM+90].

Information Alignment

Information alignment is the ability to control that the participating roles agree on the outcome of the interactions or even of the execution of the whole protocol [W3C05]. The implementing system should ensure that the specified requirements are satisfied (i.e., the interaction complete and the partner have the same information understanding, or choreography termination state is agreed).

We distinguish two kinds of properties to be modelled and validated on the implementing composition. The properties of the first group are used to check the proper interaction completion and the corresponding data alignment. The property of the second group are used to verify that the participants have a common view on the termination state.

More formally, let $a = (r_s, r_r, \mu, \bar{v}_s, \bar{v}_r) \in \mathcal{A}_O$ be an interaction action whose alignment has to be ensured. Let also ϕ be an expression over the variables of the partners that is expected to evaluate to true on the completion of the interaction. The interaction alignment rule requires that any emitted message should be eventually consumed, a new message can not be emitted until the previous is consumed, and the values of the variables should satisfy the expression on the interaction.

Definition 5.4 *An* interaction alignment rule $\langle (r_s, r_r, \mu, \bar{v}_s, \bar{v}_r), \phi \rangle$ *requires that for any run* $\omega = \gamma_0, a_0, \gamma_1, a_1, \ldots$ *of* Σ^I*, if* $\gamma_i \xrightarrow{\overrightarrow{\mu}} \gamma_{i+1}$ *for some* $i \geq 0$*, then*

- *there exists* $j > i$*, such that* $\gamma_j \xrightarrow{\overleftarrow{\mu}} \gamma_{j+1}$*, and*
- *for any* $i < k < j$ $a_k \neq \overrightarrow{\mu}$*, and*
- $\gamma_{j+1} \models \phi$*.*

The interaction alignment rule may be verified using CTL [Eme90] model checking techniques. More formally, let $IR = \langle (r_s, r_r, \mu, \bar{v}_s, \bar{v}_r), e_{IR} \rangle$ be an interaction alignment rule. Let $\phi_{\overrightarrow{\mu}}$ (respectively, $\phi_{\overleftarrow{\mu}}$) be an expression, which is true if and only if the message μ is emitted (resp. received). A CTL formula ϕ_{IR} is defined as follows:

$$\phi_{IR} = \mathrm{AG}(\phi_{\overrightarrow{\mu}} \Rightarrow ((\mathrm{AF}(\phi_{\overleftarrow{\mu}} \wedge e_{IR})) \ \wedge \ \mathrm{A}(\neg\phi_{\overrightarrow{\mu}} \mathrm{U} \phi_{\overleftarrow{\mu}}))) \ .$$

In other words, from each state, where the aligned interaction is started, (i) the state, where the interaction is complete, should be always reachable, (ii) the information alignment condition should be satisfied, and (iii) there should not be intermediate emissions.

The coordination alignment rule requires that the participants agree on the information in a termination state of the choreography. Given some termination state $\bar{s}$, let $\phi_{\bar{s}} = \phi^1_{\bar{s}} \wedge \cdots \wedge \phi^n_{\bar{s}}$ be an expression over the implementation that evaluates to true if and only if the participants are in the required state. Let E be a set of the all the expressions of the terminating states: $E = \{\phi_{\bar{s}}\}$.

Definition 5.5 *A* coordination alignment rule $E = \{\phi_{\bar{s}}\}$ *requires that*

- *for each* γ *of* Σ^I*, with* $out(\gamma) = \emptyset$*, there exists* $\phi_{\bar{s}} \in E$*, such that* $\gamma \models \phi_{\bar{s}}$*;*
- *for each* $\phi_{\bar{s}} \in E$*, there exists* γ *of* Σ^I*, such that* $out(\gamma) = \emptyset$ *and* $\gamma \models \phi_{\bar{s}}$*.*

The verification may be performed using the following CTL formula:

$$\phi_{CR} = (\mathrm{AF} \bigvee_{\bar{s}} \mathrm{AG}\phi_{\bar{s}}) \ \wedge \ (\bigwedge_{\bar{s}} \mathrm{EF}\ \mathrm{AG}\phi_{\bar{s}})\ .$$

The formula states that some of the allowed termination states is always reachable, and each of them may be reached by some execution of the composition.

5.3 Simulation-based Composition Analysis

Process algebras provide certain features for the design and verification that deal with simulation and bisimulation. These features enable specific analysis approaches that are important in the Web service domain. In fact, allowing the modularization, the modelling from a process algebra point of view supports the distributed development and the forms of analysis critical for the software reuse. In particular, the following kinds of analysis are enabled:

- bisimulation, to check whether the behaviors of two services or two versions of the same service are equivalent. This opens up a possibility for the conformance testing, service redundancy, and the service replaceability analysis in the Web service compositions [BSBM04].
- simulation, to check whether the behavior of a process is included in the behavior of other interacting processes. A hierarchical refinement design methods is based on the notion of simulation. Moreover the simulation can be part of the problem of automatic composition of services.

Hierarchical refinement [LS84, Kur94]

. It is a well-known method for design development. It proceeds top-down: starting with a highly abstract specification, we construct a sequence of behavior descriptions, each of which refers to its predecessors as a specification, and is thus less abstract than the predecessor. At each stage the current implementation is verified to satisfy its specification. The last description in the sequence contains no abstractions, and constitutes the final implementation. The behavioral equivalence between a specification and its implementation is checked by simulation or by a trace-based equivalence. The advantage of using a

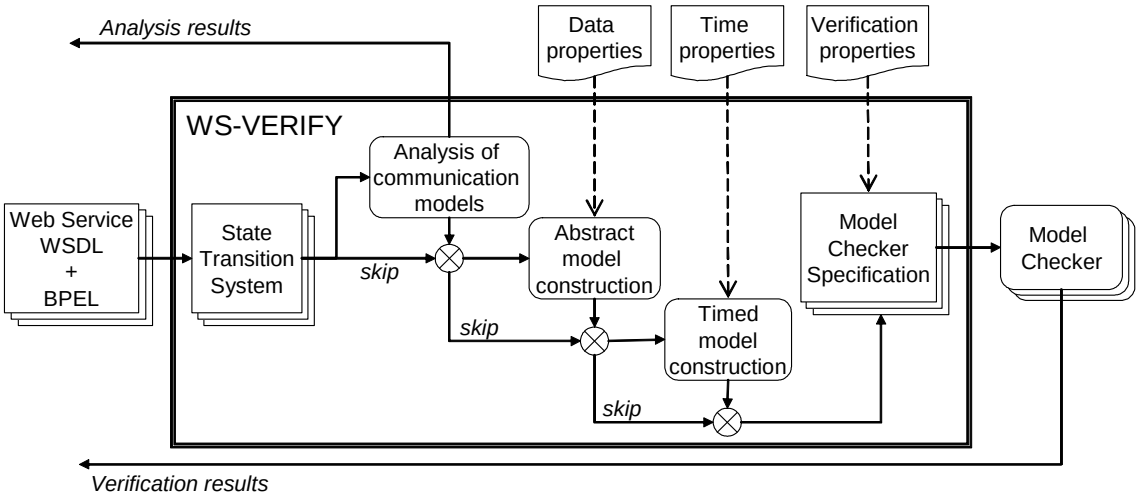

Figure 5.1: Architecture of the WS-VERIFY toolkit

two-way mapping, rather than only the direction starting from BPEL, is that we can apply hierarchical refinement also in the BPEL modelling of WS.

Automatic Composition and Redundancy. The simulation can be part of the problem of automatic composition of services: intuitively, a service is composable from a bundle of other ones, if it can be simulated by them, that is if its behaviors are contained in their behaviors. When a community of Web services is used to compose a new service (e.g. [BCG+03]), it is useful to know which services in the community are redundant: we calculate it off-line, using bisimulation. On-line, before starting the composition algorithm, we select services avoiding that two or more equivalent services are activated.

5.4 Implementation

The analysis techniques and approaches presented here are implemented and incorporated into a prototype toolkit WS-VERIFY. The architecture of the toolkit is represented in Fig. 5.1.

The specification of the Web service composition may be translated into the LOTOS models, or directly into the State Transition System format. Currently, the composition specification 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 ab-

straction 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 skipped 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 [CCGR00] and SPIN[1] [Hol97].

Alternatively, it is possible to generate LOTOS models from the BPEL/WSDL specifications and use PA analysis tools, e.g. CADP[2], for the reasoning on these specifications.

[1] The analysis/construction tasks for the SPIN model checker currently are supported only partially.

[2] `http://www.inrialpes.fr/vasy/cadp/`

Chapter 6

History of the Deliverable

The research work presented here had been carried out during the last three years of the KLASE project. The activities and results along these years are described as follows.

6.1 2nd Year

The work on the verification of Web service compositions started from an effort to describe and formalize the problem and the corresponding domain. The main focus was done on the formalism suitable for the description of the Web service compositions, and on the figuring out initial analysis tasks, such as verification of deadlocks/livelocks, requirements analysis, composition consistency.

The initial verification techniques and prototypes based on these formalisations aimed mainly at the control flow analysis, ignoring data and time aspects in the analysis process.

6.2 3rd Year

An attempt to apply the analysis techniques developed so far to the analysis of various realistic case studies immediately demonstrated both the technical and conceptual restrictions of these approaches. In particular, the complexity and diversity of interaction mechanisms used in the Web service domain, the complexity of data representation and management, the importance of handling time-related aspects of the compositions, required more accurate modelling and analysis, potentially based on new algorithms and approaches.

During the 3rd year of the KLASE project the work was devoted to the development of these techniques and methodologies focused on the specific features of Web service domain. The results of this work, consequently presented in several conferences, extend

the analysis framework with the techniques for the verification of message flow, data flow, and time flow properties.

6.3 4th Year

During the 4th year of the project we extended the analysis framework providing a methodological support for the top-down analysis of the Web service composition. The following problems were addressed:

- Formal modelling and representation of global composition models, e.g., choreographies specified in WS-CDL language;
- Analysis of realizability conditions that allow for the automated extraction of the service implementation prototypes from the global models;
- Checking of conformance between the existing services or their compositions and the global model specification.

The proposed solutions are based on the techniques developed in earlier phases. Comparing to the existing approaches, the main focus is on the specific Web service features, and on the modelling aspects crucial for the global specifications, such as information/state alignment and synchronization.

Bibliography

[ACD+03] T. Andrews, F. Curbera, H. Dolakia, J. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weeravarana. Business Process Execution Language for Web Services (version 1.1), 2003.

[AD94] Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theor. Comput. Sci.*, 1994.

[AJKL06] Rakesh Agrawal, Christopher Johnson, Jerry Kiernan, and Frank Leymann. Taming Compliance with Sarbanes-Oxley Internal Controls Using Database Technology. In *Proc. ICDE'06*, 2006.

[BCG+03] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-services that export their behavior. In *Proc. of ICSOC'03*, pages 43–58, 2003.

[BCH05] Dirk Beyer, Arindam Chakrabarti, and Thomas A. Henzinger. Web Service Interfaces. In *Proc. WWW'05*, 2005.

[BCM+90] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1990.

[BCPV04] A. Brogi, C. Canal, E. Pimentel, and A. Vallecillo. Formalizing Web Services Choreographies. In *Proc. WS-FM'04*, 2004.

[BGG+05] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Choreography and Orchestration: A Synergic Approach for System Design. In *Proc. ICSOC'05*, 2005.

[BSBM04] L. Bordeaux, G. Salaün, D. Berardi, and M. Mecella. When are 2 web services compatible? In *Proc. of VLDB-TES'04*, 2004.

[BTPT06] Fabio Barbon, Paolo Traverso, Marco Pistore, and Michele Trainotti. Run-Time Monitoring of Instances and Classes of Web Service Compositions. In *International Conference on Web Services*, 2006.

[Bus03] Christoph Bussler. *B2B Integration: Concepts and Architecture*. Springer, 2003.

[BZ83] Daniel Brand and Pitro Zafiropulo. On Communicating Finite-State Machines. *J. ACM*, 30(2):323–342, 1983.

[CCG+04] S. Chaki, E. Clarke, A. Groce, J. Ouaknine, O. Strichman, and K. Yorav. Efficient Verification of Sequential and Concurrent C Programs. In *Proc. FMSD'04*, 2004.

[CCGR00] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NUSMV: a New Symbolic Model Checker. *International Journal on Software Tools for Technology Transfer (STTT)*, 2(4), 2000.

[CCMH94] S. Campos, E. Clarke, W. Marrero, and H. Haraishi. Computing Quantitative Characteristics of Finite-State Real Time Systems. In *Proc. IEEE Real-rime systems symposium*, 1994.

[CCR91] Z. Chaochen, C.Hoare, and A. Ravn. A Calculus of Durations. In *IPL*, 1991.

[CGL94] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model Checking and Abstraction. *ACM Trans. Program. Lang. Syst.*, 16(5):1512–1542, 1994.

[CGP99] E.M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[CMS05] Enzo Colombo, John Mylopoulos, and Paola Spoletini. Modeling and Analyzing Context-Aware Composition of Services. In *Proc. ICSOC'05*, pages 198–213, 2005.

[CPT01] C. Canal, E. Pimentel, and J.M. Troya. Compatibility and Inheritance in Software Architectures. *Science of Computer Programming*, 41(2):105–138, 2001.

[EM85] H. Ehrig and B. Mahr. Fundamentals of algebraic specification 1: Equations and initial semantics. *EATCS Monographs on Theoretical Computer Science*, 6, 1985.

[Eme90] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*. Elsevier, 1990.

[FBS04] X. Fu, T. Bultan, and J. Su. Analysis of Interacting BPEL Web Services. In *Proc. WWW'04*, 2004.

[Fer04] Andrea Ferrara. Web services: a process algebra approach. In *ICSOC*, pages 242–251, 2004.

[FF95] M. Feather and S. Fickas. Requirements Monitoring in Dynamic Environment. In *Int. Conf. on Requirements Engineering*, 1995.

[FUMK03] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-Based Verification of Web Service Compositions. In *Proc. ASE'03*, 2003.

[GP03] Frank Guerin and Jeremy Pitt. Verification and Compliance Testing. In *Communication in Multiagent Systems*, pages 98–112, 2003.

[GRO00] W3C SOAP WORKING GROUP. Simple Object Access Protocol (SOAP) 1.1, 2000. [http://www.w3.org/TR/SOAP/].

[GS97] S. Graf and H. Saidi. Construction of Abstract State Graph with PVS. In *Proc. CAV'97*, 1997.

[HBCS03] R. Hull, M. Benedikt, V. Christophides, and J. Su. E-Services: A Look Bbehind the Curtain. In *Proc. International Symposium on Principles of Database Systems (PODS)*, San Diego CA, USA, June 2003. ACM Press.

[HNL02] James E. Hanson, Prabir Nandi, and David W. Levine. Conversation-enabled Web Services for Agents and e-Business. In *International Conference on Internet Computing*, pages 791–796, 2002.

[Hoa84] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1984.

[Hol97] Gerard J. Holzmann. The Model Checker SPIN. *Software Engineering*, 23(5):279–295, 1997.

[ISO89] ISO. LOTOS: a formal description technique based on the temporal ordering of observational behaviour, 1989. Technical Report 8807, International Standards Organisation.

[KBG+04] Martin Keen, JinYoung Byun, Mark Grottoli, Li Hui, Ajay Mathur, Robert Skaer, Guru Vasudeva, Gerd Watmann, and Peter Xu. *Patterns: Serial and Parallel Processes for Process Choreography and Workflow*. IBM, 2004.

[KP05] Raman Kazhamiakin and Marco Pistore. A Parametric Communication Model for the Verification of BPEL4WS Compositions. In *Proc. WS-FM'05*, 2005.

[KP06a] Raman Kazhamiakin and Marco Pistore. Analysis of realizability conditions for web service choreographies. In *Proc. FORTE'06*, 2006.

[KP06b] Raman Kazhamiakin and Marco Pistore. Choreography conformance analysis: Asynchronous communications and information alignment. In *Proc. WS-FM'06*, 2006.

[KP06c] Raman Kazhamiakin and Marco Pistore. Static Verification of Control and Data in Web Service Compositions. In *Proc. ICWS'06*, 2006.

[KPP06a] Raman Kazhamiakin, Paritosh K. Pandya, and Marco Pistore. Representation, Verification, and Computation of Timed Properties in Web Service Compositions. In *Proc. ICWS'06*, 2006.

[KPP06b] Raman Kazhamiakin, Paritosh K. Pandya, and Marco Pistore. Timed Modelling and Analysis in Web Service Compositions. In *Proc. ARES'06*, 2006.

[KPR04] Raman Kazhamiakin, Marco Pistore, and Marco Roveri. A Framework for Integrating Business Processes and Business Requirements. In *Proc. EDOC'04*, pages 9–20, 2004.

[KPS06] Raman Kazhamiakin, Marco Pistore, and Luca Santuari. Analysis of Communication Models in Web Service Compositions. In *Proc. WWW'06*, 2006.

[KS02] J. Koehler and B. Srivastava. Web Service Composition: Current Solutions and Open Problems. In *Proc. of ICAPS'03 Workshop on Planning for Web Services*, 2002.

[Kur94] R.P. Kurshan. *Computer Aided Verification of Coordinating Processes*. Princeton University Press, 1994.

[Ley01] Frank Leymann. *Web Services Flow Language (WSFL 1.0)*. IBM Academy Of Technology, 2001. [http://www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf].

[LS84] S.S. Lam and A.U. Shankar. Protocol verification via projection. *IEEE Trans. on Software Engineering*, 10:325–342, 1984.

[Mil89] R. Milner. Communication and concurrency. *International Series in Computer Science*, 1989.

[OAS93] OASIS. Organization for the Advancement of Structured Information Standards, 1993. [http://www.oasis-open.org].

[OAS03] OASIS. UDDI - Universal Description, Discovery, and Integration - Specification 3, 2003. [http://www.uddi.org].

[OAS06] OASIS. *ebXML Business Process Specification Schema, Version 2.0.3*, 2006. [http://docs.oasis-open.org/ebxml-bp/ebbp-2.0/2.0.3].

[OMG05] OMG. *Business Process Modeling Language (BPML)*, 2005. [http://www.bpmi.org].

[OYP05] Bart Orriëns, Jian Yang, and Mike P. Papazoglou. A Rule Driven Approach for Developing Adaptive Service Oriented Business Collaboration. In *Proc. ICSOC'05*, pages 61–72, 2005.

[Pan01] P.K. Pandya. Specifying and Deciding Qauntified Discrete-time Duration Calculus formulae using DCVALID. In *Proc. Real-Time Tools, RT-TOOLS'2001*, 2001.

[Pan02] P.K. Pandya. Interval Duration Logic: Expressiveness and Decidability. *Electronic Notes in Theor. Comput. Sci.*, 65(6), 2002.

[Pan04] P.K. Pandya. Finding Extremal Models of Discrete Duration Calculus Formulae Using Symbolic Search. In *Proc. AVOCS'2004*, 2004.

[Pel03] Chris Peltz. Web Services Orchestration and Choreography. *Web Services Journal*, 2003.

[PTDL] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-Oriented Computing Research Roadmap.

[SMvMC02] A. Sahai, V. Machiraju, A. van Morsel, and F. Casati. Automated SLA Monitoring for Web Services. In *Int. Workshop on Distributed Systems: Operations and Management*, 2002.

[Tha01] Satish Thatte. *XLANG - Web Services For Business Process Design*. Microsoft Corporation, 2001. [http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm].

[Tyg96] J. D. Tygar. Atomicity in Electronic Commerce. In *Proc. 15th Annual ACM Symposium on Principles of Distributed Computing (PODC'96)*, pages 8–26, 1996.

[vdA02] Wil M. P. van der Aalst. Inheritance of Interorganizational Workflows to Enable Business-to-Business. *Electronic Commerce Research*, 2(3):195–231, 2002.

[W3C01] W3C. Web Service Definition Language (WSDL 1.1), 2001. [http://www.w3.org/TR/wsdl].

[W3C02] W3C. Web Service Choreography Interface (WSCI) 1.0, 2002. [http://www.w3.org/TR/wsci].

[W3C04a] W3C. Web Services Architecture, 2004. http://www.w3.org/TR/ws-arch/.

[W3C04b] W3C. XML Schema Part 0: Primer Second Edition, 2004. W3C Recommendation, available at [http://www.w3c.org/XML/Schema].

[W3C05] W3C. *Web Services Choreography Description Language Version 1.0.*, 2005. [http://www.w3.org/TR/ws-cdl-10/].

[WM02] Andreas Wombacher and Bendick Mahleko. Finding Trading Partners to Establish Ad-hoc Business Processes. In *CoopIS/DOA/ODBASE*, pages 339–355, 2002.

[WvdADtH03] Petia Wohed, Wil M.P. van der Aalst, Marlon Dumas, and Arthur H.M. ter Hofstede. Analysis of Web Services Composition Languages: The Case of BPEL4WS. In *Proc. 22nd International Conference on Conceptual Modeling (ER'03)*, Chicago, IL, USA, October 2003. Springer Verlag.