Timed FSM strategy for optimizing web service compositions w.r.t. the quality and safety issues

Olga Kondratyeva

NNT : 2015SACLL004

THESE DE DOCTORAT
de
L' UNIVERSITE D'ETAT DE TOMSK
et de
L'UNIVERSITE PARIS-SACLAY
préparée à TELECOM SudParis

ÉCOLE DOCTORALE N°580
Sciences et technologies de l'information et de la communication

Spécialité de doctorat : Informatique

Par

Mlle Olga Kondratyeva

Stratégie basée sur les machines à états finis temporisées pour optimiser la composition de services web à l'égard de la qualité et de la sécurité

Timed FSM strategy for optimizing web service compositions w.r.t. the quality and safety issues

Thèse présentée et soutenue à Evry, le 03 décembre 2015 :

Composition du Jury :

M. Pascal Poizat	Professeur, Université Paris Ouest Nanterre la Défense	Président
M. Roland Groz	Professeur, Grenoble INP	Rapporteur
M. Sébastien Salva	Professeur, Université d'Auvergne	Rapporteur
M. David Sadek	Directeur de la recherche, Institut Mines-Télécom	Examinateur
Mme Fatiha Zaïdi	Maître de Conférences, Université Paris-Sud XI	Examinatrice
Mme Nina Yevtushenko	Professeure, Université d'État de Tomsk (Russie)	Co-directrice
Mme Ana Cavalli	Professeure, Télécom SudParis	Co-directrice

Abstract

Timed FSM strategy for optimizing web service compositions w.r.t. the quality and safety issues

by Olga KONDRATYEVA

Service-oriented architecture (SOA) together with a family of Everything-as-a-Service (XaaS) concepts nowadays are used almost everywhere, and the proper organization of collaborative activities becomes an important challenge. With the goal of bringing to the end-user safe and reliable service with the guaranteed level of quality, issues of service compositions verification and validation become of high practical and theoretical interest. In the related works, numerous models and techniques are proposed, but mostly focused on functional and non-functional issues in isolation, while integration of these parameters within unified formal framework still remains a problem to be solved – and therefore became one of the core objectives of this thesis.

In our work, we address the problems of web service composition verification and optimization with respect to functional, quality and safety properties of the composition. Finite state models are proven to be useful for testing and verification purposes as well as for service quality evaluation at each step of service development. Therefore, we propose to use the model of Finite State Machine with Timeouts (TFSM) for integrating functional service description with time-related quality and safety parameters, and suggest the extension of the model in order to adequately inherit significant nondeterminism due to the lack of observability and control over third-party component services. For the purpose of component optimization in the composition, we propose a method for deriving the largest solution containing all allowed component service implementations, based on solving TFSM parallel equation. Further, techniques for extracting restricted solutions with required properties (minimized/maximized time parameters, deadlock- and livelock-safety, similarity to the initially given component, etc.) have been proposed. In cases when the specification of a composite service is provided as a set of functional requirements, possibly, augmented with quality requirements, we propose a technique to minimize this set with respect to the component under optimization. Application of the obtained results for more efficient candidate component services discovery and binding, alongside with extending the framework for more complex distributed modes of communications, are among the future work.

Acknowledgements

I would like to express my gratitude to my supervisors Professor Ana Cavalli and Professor Nina Yevtushenko for granting me the opportunity to work with both their teams, in France and in Russia, and all their patience and support during these three years.

Besides my supervisors, I also would like to thank the Fondation Telecom for the scholarship within the Futur &Rutpure program without which financial support it would not be possible for me to have this unique opportunity of working alongside international multi-cultural team in one of the most beautiful countries in the world.

I am sincerely grateful to the reviewers of my thesis – Professor Roland Groz and Professor Sébatsien Salva – for all the time they spent on commenting my manuscript and for their insightful revisions. I also thank Professor Pascal Poizat, Fatiha Zaïdi and David Sadek for joining my thesis jury and for their consideration of my work.

In addition, I would like to thank all the colleagues from the teams of former Logiciels-Réseaux (LOR) department (currently joined to RS2M) at Telecom SudParis and Information technologies in the study of discrete structures department at Tomsk State University for continuous support and interesting discussions. In particular, I am grateful to Professor Stephane Maag for practical case studies, to Natalia Kushik for being a motivating co-author, and to Maxim Gromov for introducing me to the world of research during my undergraduate study.

On a more personal level, I would like to mention how much I appreciate the support from my family and friends. Special thanks are to my sister Evgeniya for always being there for me, to Jorge and Diego for all the coffee we drank together, to João for all the mood-enlightening pictures shared, and to Christopher the Silent Fish for watching and motivating me from the wall over my desk.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

Web services:

WS	**W**eb **S**ervice
SOA	**S**ervice-**O**riented **A**rchitecture
SOAP	**S**imple **O**bject **A**ccess **P**rotocol
XaaS	Everything-**a**s-**a**-**S**ervice
BaaS	**B**usiness-**a**s-**a**-**S**ervice
IaaS	**I**nfrastructure-**a**s-**a**-**S**ervice
SaaS	**S**oftware-**a**s-**a**-**S**ervice
PaaS	**P**latform-**a**s-**a**-**S**ervice
QoS	**Q**uality **o**f **S**ervice
QoE	**Q**uality **o**f **E**xperience
SOA	**S**ervice-**O**riented **A**rchitecture
WSDL	**W**eb **S**ervice **D**efinition **L**anguage
BPEL	**B**usiness **P**rocess **E**xecution **L**anguage
WS-CDL	**W**eb **S**ervice **C**horeography **D**escription **L**anguage
RPC	**R**emote **P**rocedure **C**all
XML	E**x**tended **M**arkup **L**anguage

Formal model:

FA	**F**inite **A**utomaton
FSM	**F**inite **S**tate **M**achine
TFSM	**F**inite **S**tate **M**achine with **T**imeouts
CUO	**C**omponent **U**nder **O**ptimization

Chapter 1

Introduction

1.1 General Context and Motivations

Nowadays web services are used almost everywhere [1], and the family of concepts of providing something "as-a-service"goes far beyond web service per se. In particular, synchronizing or even moving everyday routine and/or professional activities to clouds have already made it nearly impossible to remember life without Everything-as-a-Service (XaaS). Send an email, share a project report with colleagues, create an electronic rsvp form for a party, or design animated presentation on-line using yearly subscription instead of often over-priced in-box license - the typical examples of Software-as-a-Service (SaaS), also referred to as "on-demand software"[2]. Need to deploy an application, and want to avoid the trouble of physical maintaining of servers and data centers, thinking about operating systems and runtime - the solution is Platform-as-a-Service (PaaS) [2], e.g., Google App Engine [3]. Though, if you prefer to be in control of everything but hardware - Infrastructure-as-a-Service (IaaS) [2] is another model, and one of the classical example for IaaS are Amazon Web Services (AWS) [4, 5]. And the list of "as-a-service"concepts and models could be continued. Practically, in Web 3.0 [6] era saying that everything is a service - is not an exaggeration anymore.

The number of services increases very fast in order to allow users an easy manipulation of various online applications. Different services developed for the same purpose can be found in service repositories (see, for example, [7]). Nevertheless, the same functional properties do not mean that those web services have the same quality. Thus, in order

to efficiently select the best web service among the great number of available services it is necessary to have the adequate evaluation of the service quality.

Service-oriented architecture (SOA) is an emerging technology, gaining its popularity from a range of benefits: heterogeneity, scalability, flexibility, reusability of the components, etc. [8], becoming a basis for various internet applications such as online multimedia services, purchase and payment systems, e-government, and many others. The service can be defined as a self-contained unit of functionality, logically representing a repeatable business activity with a specified outcome [9]. Simple functionality is usually implemented as a solid entity while complex services are built up from a simpler ones, often provided by a third-party.

Considering any service realizability issues, especially for composite services, it is important to take into account not only functional aspects of components behavior and interactions, but non-functional as well. Among the components with similar functionality, the optimal choice maybe done with respect to their quality parameters. Variety of quality metrics can be assigned to services, while the most popular and widely-used are objective QoS (Quality of Service) attributes. In most papers, the quality of a given web service is defined as a set or a pattern of attributes/parameters of this service [1, 10–12]. As it is mentioned in [11], the major attributes to define the Quality of Service (QoS) are the time delay, the package loss percentage, the service access facility (the availability), the reliability, etc. All these parameters are rather objective and thus, can be evaluated automatically when a range of possible values is specified in advance for each parameter.

Standardized descriptions of service functionality with languages like WSDL, BPEL or WS-CDL, usually do not contain any quality information, though some extensions were recently proposed [8, 9, 13, 14]. The quality information about services becomes the essential part of Service Level Agreement (SLA), the service level contract between service provider and service user describing mutual responsibilities [8, 9, 15]. One of the most common features of the SLAs is a contracted delivery time, related to timed-dependent QoS parameters like response delays, mean time between failures, execution time and timeouts.

The literature survey clearly shows that more research is needed in usage of finite state models for the evaluation of the quality of web services, especially while considering complex services. Often, finite state models augmented with weights and probabilities

such as Markov chains, probabilistic automata, weighted automata, etc., allow more accurate estimating of some parameters which are important for the quality of web services. In most works, the quality of web service compositions is evaluated with respect to a number of composition types and the quality of the composite service is aggregated from the values of quality parameters for service components. Though, sometimes workflows, which are often used for representing such composition, are insufficient for the precise evaluation of the quality of the composite service. Thus, more complex finite state models such as extended finite transition systems are needed to accurately evaluate the quality of the composition when the component quality is given.

There are two main concepts of service composition that are widely discussed in the literature: orchestration and choreography. Orchestration is considered to be centralized composition with the dedicated service (called orchestrator) to manage all the interactions in the system and be a mediator between the user of the service and components involved in producing the final result. Choreography, on the contrary, is considered as distributed decentralized composition, with no dedicated service to organize collaborative work, with each involved agent knowing when and with whom to engage in interaction, and with possibility of moving the user-interaction point from one component to another according to composition goals. Different from orchestration, choreography does not describe any internal actions and internal (control) message exchanges between components, only sequences of messages that are observable from global point of view. Often [16–18] choreography is considered as a *communication protocol* to which the collaborative behavior of individual peers (component services) has to conform, and in the thesis we adapt this latter interpretation of the choreography concept, with no focus on implementation details of composition, whether it is distributed or centralized.

In this thesis, we do not go deeply in the implementation and deployment questions of choosing centralized or decentralized architecture, but rather consider choreography and orchestration being composition descriptions at different levels of abstraction: while orchestration description languages, like BPEL, already provide processes that can be executed, the choreographies represent the specification, the observable desired joint behavior of the system.

The composition mode in focus of this thesis is the parallel composition, based on the suggestion that the components of a composition communicate with each other in form

of dialogs, exchanging requests and responses, one at a time, in order to process an external request to the composition and produce an external composite response to it. Moreover, the internal communications between components can be hidden from the external viewer while describing composite behavior.

In this communication model, the orchestration architecture might be represented, for instance, as parallel composition of the orchestrator - which will be the context, that treats all external communications - and the pool of candidate component services that are being orchestrated and might be seen as an embedded component, communicating with the environment only through the context. Correlation of the parallel composition to choreography in general case is more complicated, though, the resulting composition could be considered as a result of choreography between components, while the internal dialogs-mode communications could be considered as control messages that components in choreography might exchange in order to coordinate their joint behavior. However, the mode of communication we consider cannot be directly applied to the distributed computations with multiple communications and synchronizations or simultaneously working components, neither for centralized nor decentralized architecture, and those issues are left for the future work.

The use of finite state models for dealing with composition realization issues (i.e. verifying whether the given set of components can work together according to composition specification) has proven to be perspective. Often [17–20] the specification and component behaviors are represented with variants of finite automata and finite state machines, based on which authors propose formal algorithms for checking composition implementability and even derive skeletons for candidate component services.

Though, the quality issues of service compositions are mostly treated separately from functional. Integrated consideration of both functional and nonfunctional requirements may be done based on finite state models augmented with additional parameters, similar to [21–23]. In the thesis, we assume that behavior of the components and specification of the composition are representable with Finite State Machines with Timeouts (TFSM): classical FSMs augmented with timed functions for input and output symbols to integrate service time-related quality parameters, – and show how this model can be used for verifying safety properties of service composition.

1.2 Objectives and Contributions

In the thesis we address two main questions on service composition within TFSM framework:

1. Whether the parallel composition of given components satisfies given functional and quality requirements, while providing safe communication between components?
   (a) If the composition satisfies functional, but not quality requirements, is it possible to reconsider quality requirements in such way, that the composition is able to meet them?
   (b) If the composition does not satisfy functional requirements, what part of the specified behavior can it implement? Is it possible to choose components with different quality parameters so that composition functional requirements are met?
2. Whether it is possible to optimize some of the components to ensure that composition meets quality restrictions?
   (a) Is it possible to substitute a component so that a composition conforms the specification with timed restrictions?
   (b) ...without livelocks and deadlocks? (safety requirements)
   (c) ...with optimal delays? (quality requirements)

Therefore, we are stating the following objectives for the thesis:

1. Propose a finite state model capable of integrating functional, quality and safety requirements for complex services.
2. Define a composition for the proposed model for describing communicative behavior of complex services.
3. Develop methods for effective optimal component selection based on proposed model.

The work performed to fulfill the stated objectives resulted into the following contributions:

1. A FSM with Timeouts (TFSM) model has been proposed as a model for service description that integrates functional, quality-related and safety parameters.

   The FSM with Timeouts model is an extension of classical FSMs with timeout and output delays functions, therefore allowing to describe as sequences of request/response aspects of service behavior, as time-related quality properties like performance within the same model. The behavior of a component service involved in composition can be significantly nondeterministic due to possible interactions with other users and services, which we do not observe nor control, hence, in order to adequately inherit such nondeterminism in the model, an extension of the TFSM model has been also proposed.

2. A correspondence between functional conformance relations for service modeled by TFSMs in cases of real and integer-valued time variable has been established.

   In communicating service components, time variable may have any real value, which is adapted in the proposed TFSM model. Though, checking conformance between two models becomes a complex issue. Therefore, we prove that two given TFSMs are conforming while time variable has any possible real value if and only if they are in conformance while time variable has only integer values. The established correspondence allows adaptation of formal language approaches for composition and component selection issues within TFSM framework.

3. A method for deriving the largest solution containing all allowed component service implementations has been proposed based on solving TFSM parallel equation.

   It is also stated that if the solution to corresponding TFSM equation does not exist, then the composition cannot be implemented with given functional and/or quality requirements.

4. A method for minimization of the set of composite service requirements has been proposed.

   In cases when the specification of a composite service is provided as a set of functional requirements, possibly, augmented with quality requirements, this set can be minimized with respect to the component under optimization. The minimization

technique is based on determining which requirements are applied to the context only, and comparing which parts of the component under optimization are involved in satisfying or violating given requirements.

5. Techniques for extracting restricted solutions with required properties have been proposed.

   Namely, we consider four most interesting component service optimization issues:

   - selection of safe component service with respect to livelock-free composition;
   - selection of the component service with minimal number of states;
   - selection of the component service with minimal (best) or maximal (least restrictive) output delays as service quality parameter.
   - selection of the component service with minimal required changes to be made comparing with the initially given component.

The theoretical contributions are of importance not only in composite service quality evaluation and optimization domain, but also in domain of automata theory. The applications of theoretical contributions are illustrated on examples of composite services throughout the thesis.

1.3 Thesis plan

In accordance with the above objectives and to provide logical proof of contributions, the thesis is organized in 7 chapters, including Introduction (Chapter 1) and Conclusions (Chapter 7):

- Chapter 2 is devoted to the state-of-the-art in web service quality evaluation models and optimization throughout the service development steps.
- Chapter 3 focuses on the TFSM model definitions and properties, as a formal model for describing web service functional and non-functional properties.
- In Chapter 4 we discuss the correlation between web services composition concepts and the parallel composition, and formally define the parallel composition over TFSMs, providing the algorithms to derive the safety-aware composition.

- In Chapter 5 the general procedure for design of the component under optimization is provided, based on solving the parallel equation over TFSM.
- The restricted solutions correlating to the particularly interesting cases while optimizing service component are discussed in Chapter 6.
- In Conclusions (Chapter 7) we summarize the contributions and discuss the directions for future work.

1.4 List of publications

Journal papers

1. Kondratyeva O., Kushik N., Cavalli A., Yevtushenko N. Using Finite State Models for Quality Evaluation at Web Service Development Steps. *International Journal on Service Computing (IJSC)*, ISSN 2330-4472, 2013. Issue 1(1), pp. 1-12.
2. Kondratyeva O., Yevtushenko N., Cavalli A., Solving parallel equations for Finite State Machines with Timeouts. *Proceedings of the Institute for System Programming.* 2014. Volume 26 (Issue 6), P. 85-98 (peer-review journal, in Russian)
3. Kondratyeva O., Yevtushenko N., Cavalli A., Parallel composition of nondeterministic Finite State Machines with Timeouts. *Tomsk State University Journal of Control and Computer Science.* 2014. Volume 2 (Issue 27). P. 73–81. (peer-review journal, in Russian)

Conference publications

1. Kondratyeva O., Kushik N., Cavalli A., Yevtushenko N. Evaluating Quality of Web Services: a Short Survey / Proceedings of the IEEE 20th International Conference on Web Services (ICWS 2013), July 2013. – pp. 587-594. (rank A conference)
2. Kondratyeva O., Kushik N., Cavalli A., Yevtushenko N. Evaluating Web Service Quality using Finite State Models / Proceedings of the 13th International Conference on Quality Software (QSIC 2013), July 2013. – pp. 95-102.

Participation in seminars and conferences

1. Poster presentation at La Journée Futur & Ruptures (l'Institut Mines-Télécom), the best poster award, January 2014
2. International Conference on Web Services (ICWS 2013), 27 June – 2 July 2013, Santa Clara, California, USA (rank A conference)
3. First Franco-Russian Seminar on Software Verification, Testing, and Quality Estimation, November 2014, Paris, France
4. 10th International Summer School on Training And Research On Testing (TAROT 2014), 30 June – 04 July 2014, Porto, Portugal
5. Tarragona International Summer School on Trends in Computing (SSTiC 2014), 07 July – 11 July 2014, Tarragona, Spain

Chapter 2

Background for the State Models in Service Quality Evaluation

2.1 Introduction

We start the chapter with the general preliminaries and the context for web services and service quality evaluation, and continue with the survey of the formal models useful for the quality evaluation at each step of service development process.

The development steps we consider are inspired by [24] with some modifications according to quality evaluation goals at each step. Namely, the steps are: service requirements specification, provisioning, composition, implementation, and usage and management.

The step we are most interested in is the service composition, including issues of composition realizability and functional and non-functional optimization of the composition via component quality-aware selection and substitution.

2.2 Preliminaries on web services

2.2.1 Service-oriented architecture

Service-oriented architecture (SOA) is an emerging technology, gaining its popularity from a range of benefits: heterogeneity, scalability, flexibility, reusability of the components, etc. [8], becoming a basic for various internet applications such as online multimedia and OTT services, online purchase and payment systems, e-government, and many others.

The core unit of SOA is service, which can be defined from different points of view. The Open Group [9] defines the service as a self-contained logical representation of a repeatable business activity, which may be composed of other activities, given to consumers as a "black box" and has a specified outcome. Another way of defining a service is as a composition of web applications where a server (client) in one application can be turned into a client (server) in another application, for example, in [8] a service is defined as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically, WSDL). Other systems interact with the service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards".

The Open Group consortium [9] defines SOA as an architectural style supporting service-orientation, i.e., it is comprised of the following principles [25]:

- *Standardized Service Description.*
- *Service Abstraction*: service descriptions only contain essential information; implementation details are not provided to service consumer.
- *Service Reusability*: services can be positioned as reusable enterprise resources.
- *Service Discoverability*: services are supplemented with communicative meta data by which they can be effectively discovered and interpreted.
- *Service Composability*: services are effective composition participants, regardless of the size and complexity of the composition:

The communication of simple component services may be organized via a remote procedure call (RPC technology), though, the Simple Object Access Protocol (SOAP) or Representational State Transfer (REST) are more often used. In order to support main features of the heterogeneous service oriented architecture (SOA), the interaction with services is processed through public interfaces that are often defined and described using XML-based languages.

The *service description* should contain the description of service semantics and a machine-processable description of the messages that are processed by the service [8]. One of the basic standard languages is the web service description language (WSDL). A number of service depositories publish and provide the automatic analysis of WSDL service files; for example, in [26], a "WSDL Analyzer" extracts the list of supported operations and a required transport protocol that usually is HTTP. Complex services may be derived as a composition of simpler ones, and special languages for the composite service description are developed.

The simple functionality is usually implemented as a solid entity while complex services are built up from simpler ones, often provided by a third-party. Therefore, the properties of the services involved in the design of a new composite service must be carefully evaluated, and before engaging a service component into the desired collaboration, it should be verified whether it would be able to provide the required compositional behavior or not. Workflows which support the logic execution of composite complex services can be described using the business process execution language (BPEL) that, for each component service, defines which messages it gets from and sends to other components. The sequences of the message exchange occurred in the system are usually described using the web service choreography description language (WS-CDL) [20]. Canonical descriptions of services allow an automatic search for a service with the required functionality, though, the information on the quality of a selected service is usually not presented.

2.2.2 Service quality parameters

In order to efficiently select the best service among the great number of available services which can provide the required functionality it is necessary to have the adequate evaluation of the service quality.

In most papers, the quality of a given service is defined by a set or a pattern of attributes/parameters of this service [1, 10–12].

The QoS (Quality of Service) can be defined by a set of attributes (or parameters), such as the response time, availability, reliability, etc., corresponding to a given web service and allowing to compare and rank services with similar functionalities [1, 10–12]. This set of attributes is often mapped into a single value, quality score, using an appropriate computable function and the result of this function can be an integer, a rational, a (fuzzy) logic constant, etc. The QoS parameters are rather objective and thus, can be evaluated automatically when a set of possible values is specified in advance for each parameter.

For estimating the quality of web services based on the above parameters heuristic methods are usually proposed [10, 27]. A usual way when evaluating the QoS based on traffic analysis is to use a linear combination of weighted network parameters [12]. Sometimes coefficients of the formula are not given as they can be a know-how of a company that evaluates the service quality for its own purposes. For many services that are located in repositories the QoS is evaluated based on parameters mostly related to traffic analysis. However, the parameters used in traffic analysis are more concerned about the transport level than about service issues; thus, new parameters appear that have to be considered when evaluating the service quality. Such parameters may be the amount of money to be spent, service reputation, the comfort of a solution proposed by the service, etc. A single utility function that maps the values of all parameters into a single resulting value still is used for QoS evaluation [28] but usually it is difficult to represent all these parameters by means of a single value. Correspondingly, the service quality is represented as a set or a vector of heterogeneous attributes [28–30]. The parameters which are related to the traffic analysis are usually evaluated based on the traffic monitoring but such parameters as the comfort of a proposed solution, money to be spent etc. can hardly be estimated applying traffic based methods and thus, there is a need for more complex models to be involved. Taking into account sequences of various user requests, cancellations etc., web services are often described by the sets of permissible sequences of actions [1, 31] and for this reason, researchers turn their attention to trace models such as flow graphs, Markov chains, weighted and probabilistic automata, Petri nets, fuzzy logic [21, 28, 32], etc.

2.2.3 Service composition realizability and optimization

In this section, we consider in more details specifications of composite services, on different abstraction levels, and discuss whether two main composition concepts (namely, orchestration and choreography) are indeed so different.

Choreographies are becoming one of the most powerful, scalable and flexible way of describing collaboration of heterogeneous distributed components, varying its applications from service compositions [33] to complex business processes [34, 35], and far beyond, from ubiquitous systems [36] to Internet of Things [37] and coordination of mobile devices, people and offline services with online ones [38].

One of the problems of components involved in choreographies is that they cannot be forced to proceed in one or another way, unlike when having an orchestrator-manager which is eligible for making any decisions. But in real-world compositions it is not always possible to design an appropriate orchestrator, for various reasons: involved services should preserve the control of interaction with the user to themselves for security and privacy reasons (e.g., payment services like PayPal, Verified-by-Visa, WebMoney, etc., must never share with sellers the details of clients payment information), or physical impossibility (airport timetable service cannot gain any control over the planes and the air companies crews), etc. These type of compositions can be efficiently treated as choreographies, moreover, choreographies with partial specifications requiring from involved components to be able to support desired conversations/interactions but not being able to force them to do it, or ensuring that whatever components do they would never cause undesirable behavior of composition, or verifying that the components can be involved only in specified allowed conversations and none of the others.

The matter of distinguishing choreographies and orchestrations in complex services description is sometimes related not only to the degree of distribution of the control over process among involved components, but also to the level of abstraction at which the system behavior is considered.

Considering any service realization issues, especially for composite services, it is important to fulfill not only functional aspects of components behavior and interactions, but non-functional as well. Among the components with similar functionality the optimal

choice maybe done with respect to their quality parameters. Variety of quality metrics can be assigned to services, while the most popular and widely-used are objective parameters-related QoS (Quality of Service) attributes, subjective evaluation of end-user satisfaction with QoE (Quality of Experience) and revenue-related QoBiz (Quality of Business) [8]. Standardized descriptions of service functionality with languages like WSDL, BPEL or WS-CDL, usually do not contain any quality information, though some extensions were recently proposed [8, 9, 13]. The quality information about services becomes the essential part of Service Level Agreement (SLA), the service level contract between service provider and service user describing mutual responsibilities [8, 9, 15]. One of the most common features of the SLAs is a contracted delivery time, related to timed-dependent QoS parameters like response delays, mean time between failures, execution time and timeouts.

There are two main issues with service composition that can be addressed within formal models frameworks:

- Given a composition specification (most likely, choreography), whether the composition of given components conforms specification? If yes, can the composition of given components satisfy all the quality restrictions? (composition realizability issue)
- Given a composition specification and the components that conform to it, whether it is possible to optimize some of the components to ensure that the composition meets quality restrictions? What restrictions should be put to components so that they satisfy both functional and quality requirements of the composite service? (composition quality- and safety-aware optimization issue)

In related work, the questions of composition realizability are handled within appropriate finite automata-like models, like component automata [17], conversation protocols [16], symbolic transition systems [39], and the usual way of deriving the components from the composition specification is to perform natural projection [17] and/or augment the result of the projection with special control messages to ensure that involved components are aware of the nondeterministic choices made by other components [18]. In [18] it is also

shown that realization of given choreography can be done either in centralized or decentralized manner, using special control messages in both cases to overcome drawbacks of simple natural projection and allow realization of more challenging choreographies.

In most cases, the models used to analyze the realizability of given composition are based on finite state models augmented with corresponding communication operators.

2.3 Evaluating quality of web services at development steps using finite state models

2.3.1 Web service development steps

As any software product, a web service passes different steps while being developed, and these major steps can be defined in different ways: starting from classical software development life cycle models [40, 41] to business process development [42, 43]. Whatever chosen development steps are, evaluating quality of the service under development and verifying its conformance to functional requirements at each step are important issues for more flexible modification and re-design of either service or requirements. We consider a chain of five development steps inspired by [24] and slightly modified to be better adapted for the quality-aware service development, namely, service requirements specification, provisioning, composition, implementation, and usage and management (Fig. 2.1). An implementation step is explicitly distinguished and the usage and management steps are united as they are closely related when evaluating the quality.

1. *Service requirements specification*: at this step, functional and non-functional requirements for a service under construction are set, the service and interface descriptions are specified; in particular, a formal service model can be derived.

2. *Service provisioning*: this step is used for estimating required and/or available provider and business resources.

3. *Composition of services*: at this step, a service under construction is decomposed into simpler services. A decision is made on already implemented component services that could be selected in order to guarantee the necessary functionality and quality of the composite service.

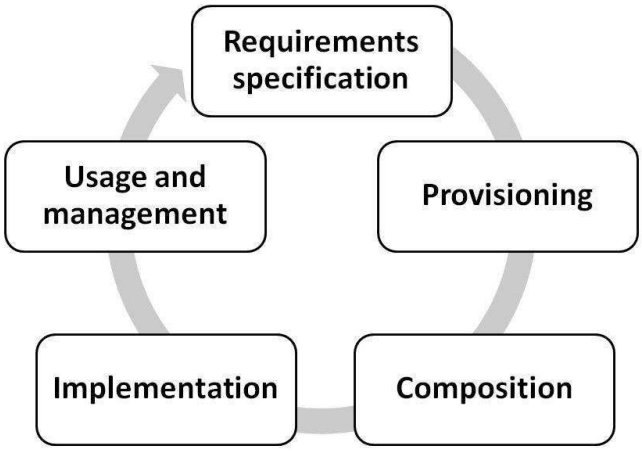

FIGURE 2.1: Web service development steps

4. *Service implementation*: at this step, implemented components of a composite service are combined together; verification and testing of the new service are also performed.

5. *Service usage and management*: as the main goal of each service is to satisfy an end-user, the quality of a developed service should be monitored during its usage by collecting and analyzing corresponding data. If the provided quality is under the given threshold then there can be a need for the service optimization.

Service quality can be eventually improved/deteriorated at each development step. Generally, this does not guarantee the optimum of the final service quality but usually helps to get closer to the target. For this reason, a quality evaluation is performed at each service development step. At the first two steps, the specification and provisioning steps, a set of service requirements as well as a set of available resources needed for the service implementation are determined.

Example 2.1. *Throughout the section we illustrate the use of finite state models at different development steps for the quality evaluation with the example of a* Vacation Planner *service. This example, with slight modifications, is taken from [44, 45]. The service allows a user to purchase flight tickets and to book an accommodation at the destination point. A user submits traveling dates and the planner proposes a number of available options for flight tickets and hotel rooms. If the user and planner agree on the flight ticket and hotel room then the vacation is successfully booked. Otherwise, the vacation reservation has failed.* △

2.3.2 Specifying service requirements and deriving a formal specification

The first step - the service specification - has the following goal: describe what the service under development should be able to do and what is the expected quality of the service we hope to be able to provide.

The requirements for the service under construction can be implicitly divided into two groups: functional and non-functional requirements. At the first step it is important to specify which quality parameters are crucial for the service quality and this choice essentially depends on web service features. The choice of quality parameters has a significant impact on the further attractiveness of the constructed service.

The list of crucial parameters is usually defined as the quality vector $QoS = \langle q_1, q_2, \ldots, q_n \rangle$. The restrictions on parameter values (if there are any) can be, for instance, stated by a system of equations or inequalities. Equivalence and order relations can be used for specifying the priorities over quality parameters [46]. For example, the notations below can be used:

- $q_i \succ q_j$ – parameter q_i is more important than parameter q_j;
- $q_i \succ\succ q_j$ – parameter q_i is *much more* important than parameter q_j;
- $q_i \approx q_j$ – parameters q_i and q_j have the same importance.

To be able to produce a "better" service than those which already exist, it is necessary to define the notion "to be better" for two services. Two most popular approaches for such evaluation are the usage of a utility function and the Pareto-dominance relationship.

Utility function [28, 29, 47] is a computable function that maps the QoS vector into a single quality score $F(QoS)$. In other words, given two services S_1 and S_2 with the quality vectors QoS_1 and QoS_2, the service S_1 is *better* than S_2 if $F(QoS_1) > F(QoS_2)$. There are many ways for defining utility functions; the simplest option is to define such a function as a weighted sum of quality parameter values, $F(QoS) = \sum_{q_i \in QoS} \omega_i q_i$ where the main question is how to choose weights. In [28], weights are chosen in such a way that a weighted parameter value is between 0 and 1 where 0 corresponds to the "worst" parameter value while 1 corresponds to the "best". In [46], the weights are calculated

based on a partial order relation between parameters. The "worst" and "best" parameter values essentially depend on a parameter. For instance, for the response time the less the value is, the better it is for the service quality, while for the availability or popularity a better value is the greater one.

The utility function is not always appropriate for the service quality comparison, especially for composite services. Last years, the *Pareto-dominance* relationship, also often called *skyline*, is increasingly used [46, 48]. Once the relation "better" is set for the set of values of each parameter, the quality vector QoS_1 is said to *dominate* vector QoS_2 if all components of QoS_1 are "not worse" than corresponding components of QoS_2 and at least one component of QoS_1 is better than that of QoS_2. The importance of parameters may be easily taken into account, by considering the domination only over most important parameters. Less important parameters are not taken into consideration at all or the maximal discrepancy value can be set for these parameters.

As web services often process permissible sequences of actions, finite state models are widely used for their analysis and synthesis. In [19], the authors extract a Finite State Machine (FSM) from the WS-CDL description using the tool DIEGO 2.0. In [20], the BPEL and WS-CDL descriptions are translated into a system of communicating timed automata. In most papers, the extracted finite state models are used for automatic composition of services [49] or testing and verification purposes [50, 51]. In the next sections, we show that finite state models can also be helpful for more precise quality evaluation.

Example 2.2. *For the vacation planner informally described in Example 2.1, we set the following functional requirements: 1) Vacation is booked if both a flight ticket and a hotel room are reserved, 2) Flight tickets are proposed before booking a hotel room. Quality requirement at this step can be expressed with a restriction, that each user request has to be processed in at most 30 seconds.*

For the formal specification, on a high level of abstraction, the functional requirements can be represented by transitions of a finite automaton in Fig. 2.2.

The service starts with waiting in the initial state q_0 for users request on preferable travel dates, replying in the state q_1 with the dates available for flight tickets. Then the user might agree or disagree with proposed options (moving from state q_2 to q_3). In case of

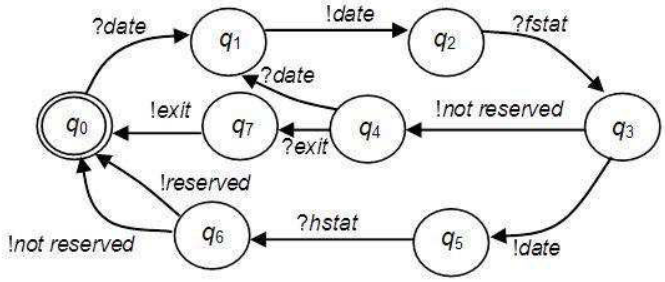

FIGURE 2.2: Finite automaton model for the Vacation Planner service

disagreement, the service moves to state q_4 and proposes user to either check other travel dates and continue (back to state q_1) or quit the service via state q_7.

If the user accepts the ticket provided by the service at state q_3, then the service confirms available dates and options for booking a hotel room, moves to state q_5 and waits for user to decide on hotel reservation. After receiving the answer from the user, the service quites to the final state (coinciding with the initial one) with confirmation or cancellation of the travel booking (with corresponding outputs !reserved or !not_reserved). △

2.3.3 Estimating reachable quality of the service under development

The service provisioning step includes the analysis of available and required resources for the service under development. This issue involves business processes and related resources such as human-hours to be spent, the cost of implementation of a new service comparing to the usage of existing components, etc. All these parameters significantly influence the system architecture as well as a choice of component services and the implementation quality but their influence is rather implicit and for this reason, such business issues are left out of the scope of this thesis. Nevertheless, knowing some network resources (server quality, internet speed, etc.) gives a chance for more precise evaluation of some service parameters such as the response time or availability of the service. The marketing analysis of the target audience can also refine the evaluation. The quality evaluation at this step can become a part of Service Level Agreement (SLA) [52]. If the predicted quality is under desirable standards then the service requirements specification should be revised, i.e., a developer should come back to the previous development step.

Given a finite state model extracted from some service description, at the provisioning step the model can be refined by augmenting its states and/or transitions with

weights, probabilities or other attributes based on the quality requirements. In particular, in [21], weighted automata are used for more precise service quality evaluation. The weight associated with each automaton transition represents the cost of the corresponding transaction execution (in terms of time, money, etc). The quality parameter values can be estimated via different execution paths of the corresponding automaton and some conclusions about the service quality can be drawn.

Example 2.3. *Continuing with the Vacation Planner service example, we illustrate the benefits of using weighted automata comparing to the simple rough estimation. For the purpose, the automaton from Fig. 2.2 is augmented with weights for transitions corresponding to service transactions, weights representing the estimated maximal time to perform each operation. The transitions corresponding to the actions of the user (input actions marked with "?") are not weighted because the user might take as much time as he/she needs to perform a chosen operation, and hence are not taking into account for evaluation of service quality. The resulting automaton is shown in Fig. 2.3.*

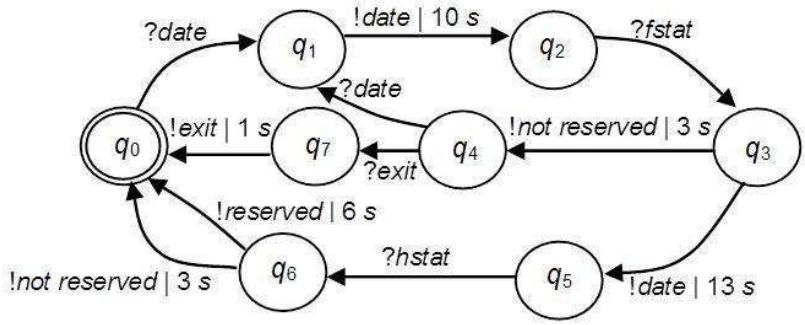

FIGURE 2.3: Weighted automaton for the Vacation Planner service

The rough quality estimation that does not take into account the order of operations (and corresponding paths), could be done by simple sum of all known transaction delays, which in our example return 36 seconds and violates the quality requirement described for the vacation planner in Example 2.2.

Though, if the estimation is done based on the formal model, weighted automaton in this case, the more accurate value may be obtained by considering traces from the initial to the final state. As mentioned in [21], the weight of each simple trace (without cycles) may be calculated and the longest transaction time corresponds to a trace with the maximal weight - in this example, such trace is the one through sequences states $q_0q_1q_2q_3q_5q_6q_0$ *with total weight 29 seconds.* △

2.3.4 Service composition and implementation

When a new service is derived by composing already existing services, possibly developed by a third-party, two main questions arise: what service quality can be provided by existing components, and how to select components to achieve a higher quality of composite service.

For quality evaluation purposes, the functioning of the component services and the content of the messages they exchange are often not considered. When designing the architecture of a composite service, task invocations between components can be expressed with a corresponding workflow and Fig. 2.4 illustrates the basic composition patterns in which the workflow can be decomposed. When a component service is invoked, it executes some task (according to the composition requirements), and after completing the task, the service either produces the result if it is the final task, or invokes other components to execute further composition tasks. The same component service can be used for executing different composition tasks. To avoid any ambiguity further the execution of some task by the service is referred to as a component service.

The simplest workflow compositional pattern is sequential (Fig. 2.4a) where the composite service is organized as follows: when the service S_0 completes a task then the service S_1 is invoked. In a conditional pattern (Fig. 2.4b), also referred to as XOR-split pattern, S_0 invokes one and only one of services $S_1, \ldots, S_k$ depending on the results of the task execution. When probabilities are involved for each possible invocation, the equality $\sum_{i=0}^{k} p_i = 1$ must be held. When the service S_0 invokes several services $S_1, \ldots, S_k$ a parallel pattern (Fig. 2.4c), or an AND-split pattern, is considered. Services executing tasks in parallel can be further merged in order to execute a required later task. If the next service is invoked only when all preceding services have completed their tasks, the synchronizing pattern (Fig. 2.4d), or AND-joint pattern, is considered. Otherwise, if the next service is invoked after at least one service completes its task, a concurrent pattern (Fig. 2.4e), or a XOR-joint pattern, is at hand. When some tasks should be repeated, the loop pattern (Fig. 2.4f) is involved, and the number of repetitions may be either known a priori or can be calculated during the task execution.

Without loss of generality, loops are considered to have a single starting point (service S_1 in Fig. 2.4f) and any number of exits (service S_i in Fig. 2.4f). For each exit point, the probabilities of continuing the loop or of going out may be specified.

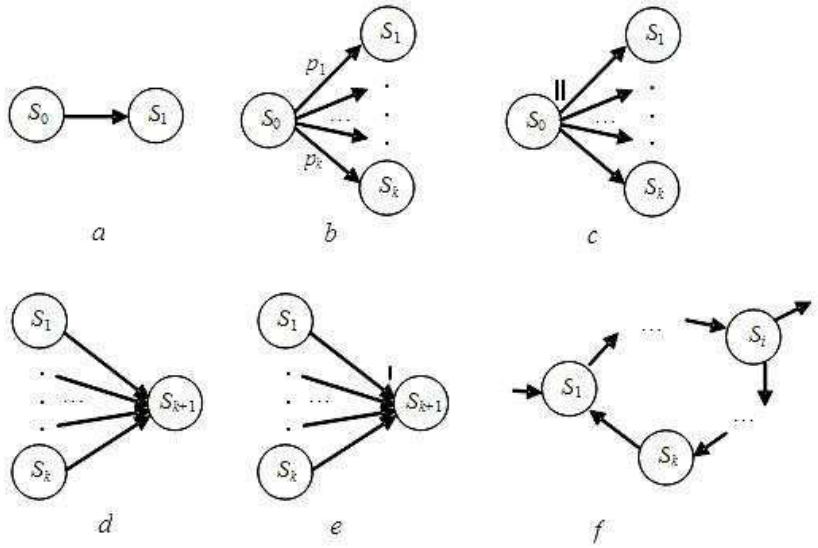

FIGURE 2.4: Basic compositional patterns: (a) sequential, (b) conditional, (c) parallel, (d) synchronizing, (e) concurrent, (f) loop

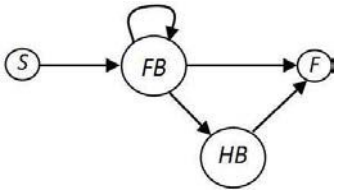

FIGURE 2.5: Workflow for the Vacation Planner service

Example 2.4. *For the Vacation Planner service (Examples 2.1 to 2.3) the workflow consists of two components - Flight Booking (FB) and Hotel Booking (HB) services, which are invoked sequentially, and the Flight Booking may be re-invoked repeatedly in a loop, as shown in Fig. 2.5.* △

2.3.4.1 Aggregation functions for quality evaluation

Given a set $\{S_1, S_2, \ldots, S_n\}$ of component services, their QoS vectors $\{Q_1, Q_2, \ldots, Q_n\}$ (for instance, published by service providers or so-called service brokers [53], and the composition structure, the question is: what is the quality Q of the composite service?

For all basic patterns and their combinations the question of overall QoS evaluation has been studied properly and aggregation functions have been elaborated for a number of QoS parameters. The result of each aggregation function is the value of a corresponding attribute of the composite service. Some of these functions without considering probabilities for XOR-splits are summarized in Table 2.1. In Table 2.2, probabilities for

TABLE 2.1: Aggregation functions for quality parameters

QoS Attribute	**Compositional pattern**					**References**
	Sequential	XOR-split	AND-joint	XOR-joint	Loop	
Cost (c)	$\sum_{i=1}^{k} c_i$	$\sum_{i=1}^{k} c_i$	$\sum_{i=1}^{k} c_i$	$\sum_{i=1}^{k} c_i$	$n \sum_{i=1}^{k} c_i$	[29, 53]
Availability (a)	$\prod_{i=0}^{k} a_i$	$\prod_{i=0}^{k} a_i$	$\prod_{i=0}^{k} a_i$	$1 - \prod_{i=0}^{k} (1 - a_i)$	$\left(\prod_{i=0}^{k} a_i\right)^n$	[28, 53]
Response time (t)	$\sum_{i=1}^{k} t_i$	$\max(t_i)$	$\max(t_i)$	$\min(t_i)$	$n \sum_{i=1}^{k} t_i$	[29, 53]
Reliability (r)	$\prod_{i=0}^{k} r_i$	$\prod_{i=0}^{k} r_i$	$\prod_{i=0}^{k} r_i$	$1 - \prod_{i=0}^{k} (1 - r_i)$	$\left(\prod_{i=0}^{k} r_i\right)^n$	[28, 30, 53]
Reputation (q)	$\frac{1}{k} \sum_{i=1}^{k} q_i$	$\frac{1}{k} \sum_{i=1}^{k} q_i$	$\frac{1}{k} \sum_{i=1}^{k} q_i$	$\frac{1}{k} \sum_{i=1}^{k} q_i$	$\frac{1}{k} \sum_{i=1}^{k} q_i$	[28, 30]

TABLE 2.2: Aggregation functions for quality parameters when probabilities are involved

QoS Attribute	**Compositional pattern**		**References**
	XOR-split	**Loop**	
Cost (c)	$\sum_{i=1}^{k} p_i c_i$	$\sum_{j=1}^{k} \frac{\left(\prod_{i=0}^{j-1} p_i\right)\left(1-p_j\right)\left(\sum_{i=1}^{j} c_i + \prod_{i=1}^{k} p_i \sum_{i=j+1}^{k} c_i\right)}{\left(1-\prod_{i-1}^{k} p_i\right)^2}$	[29, 53]
Availability (a)	$\sum_{i=1}^{k} p_i a_i$	$\sum_{j=1}^{k} \frac{\left(\prod_{i=0}^{j-1} p_i\right)\left(1-p_j\right)\prod_{i=1}^{j} a_i}{1-\prod_{i-1}^{k} p_i a_i}$	[29]
Response time (t)	$\sum_{i=1}^{k} p_i t_i$	$\sum_{j=1}^{k} \frac{\left(\prod_{i=0}^{j-1} p_i\right)\left(1-p_j\right)\left(\sum_{i=1}^{j} t_i + \prod_{i=1}^{k} p_i \sum_{i=j+1}^{k} t_i\right)}{\left(1-\prod_{i-1}^{k} p_i\right)^2}$	[29, 53]

XOR-splits and loops are taken into account when deriving aggregation functions. In these tables, the integer k denotes the number of involved services while the integer n is used for the number of loop iterations, and p_i is the probability of invoking the service S_i.

In fact, the set of functions in Table 2.1 is incomplete, since more attributes can be considered such as popularity, especially for social networks [54]. When involving some logic models, such as fuzzy logic, k value logic etc., or modular arithmetic models, aggregation functions which use only sum and multiplication operators do not seem to be sufficient and thus, novel corresponding aggregation functions have to be elaborated.

Table 2.2 contains no aggregation functions for sequential and parallel patterns since they coincide with those in Table 2.1. For a conditional pattern the average quality evaluation is calculated, though the quality often is computed for each path separately, which allows to assess the worst case, the best case, and/or the quality along the most probable execution path [53]. When probabilities are given, the quality of a loop pattern is calculated for an arbitrary (not limited) number of iterations.

However, sometimes workflows, which are often used for calculating the composition QoS, are insufficient for the precise evaluation of the quality of the composite service.

Thus, more complex trace models such as extended finite transition systems are needed to accurately evaluate the quality of the composition when service component qualities are given.

2.3.4.2 Quality-aware component selection

When the estimated quality of a composite service is unsatisfactory the question arises how the quality could be enhanced. According to the previous section, the quality of a composite service significantly depends on the composition structure and the quality of component services. The composition structure is mainly pre-determined by functional requirements and hence, cannot be easily changed. Correspondingly, in order to enhance the quality of a composite service there is an option of selecting a "better" component service, a so-called the quality-aware component selection problem. A "better" component service can be selected from a collection of services with similar functionalities based on a corresponding utility function or a skyline order relation.

The quality-aware component selection is a multi-dimensional optimization problem and often can be reduced to the well-known combinatorial problems such as the multi-choice knapsack problem [55], the resource constraint project scheduling [56], or the derivation of a shortest graph path under some constraints [57], etc. Local and/or global approaches can be applied [28, 55, 58]. Local selection approaches focus on choosing the best component for each task independently of other tasks, while the global optimization objective is to derive a composition with a better overall quality. For partially specified or derived on-the-fly compositions a local selection approach is reasonable, though, it cannot guarantee that the composite service satisfies the given quality requirements [59]. Global approaches which can ensure the composite service quality utilize the multi-dimensional optimization [58–60] and these approaches are rather computationally expensive. Novel promising approaches are rather the mixture of local and global approaches [28, 46, 48].

2.3.4.3 The implementation issues

When all decisions are taken about the service hierarchy and components, the non-existing component services and the overall composition are implemented. At this step, formal descriptions using finite state models can essentially help as there are many tools

Service Level	Description	Response Time	Solution Time
Gold	Premier support service. Recommended for business critical environments.	1 hour	1 day
Silver	Recommended for non-critical machines that are used for development & testing purposes or low-risk production environments.	4 hours	3 days
Bronze	Recommended for low-risk machine environments that are used for irregular activities (e.g., ad-hoc Research & Development).	1 day	1 week

FIGURE 2.6: An example of response time SLA restrictions [63]

which allow automatic code generation (see, for example, [61]). A developed implementation is then verified and tested for checking that the implementation conforms to its specification [50, 51, 62]. Other properties of the implementation such as security and robustness can be also tested at this level. Nevertheless, the quality related features which are expected according to the service specification usually are tested at the next step when the service is started to be used. If some parameter values do not satisfy a developer he/she can come back to the previous steps in order to redesign the service.

2.3.5 Service usage and management

The main objectives of service management include, but are not restricted to, improving service quality and ensuring that the service satisfies functional and non-functional requirements [8, 15]. Requirements that service should meet can be specified in the Service Level Agreement [52] and are checked via service monitoring when various data are collected and analyzed. At this step, detecting the SLA violations becomes one of the main service management objectives.

The quality information about services becomes the essential part of Service Level Agreement (SLA), the service level contract between service provider and service user describing mutual responsibilities [8, 9, 15]. One of the most common features of the SLAs is a contracted delivery time, related to timed-dependent QoS parameters like response delays, mean time between failures, execution time and timeouts. In this paper we focus on those parameters, integrating them into the state model of service behavior in terms of dedicated timed functions. The example of timed-related fragment of an SLA document for a simple IT helpdesk service from [63] is shown in Fig. 2.6.

Monitoring the conformance of service performance to the specification of composition and SLA requirements is one of the crucial objective at this development step. In cases

if specification and/or SLA violations are detected, the optimization of the service, including substitution of some components, might be required.

2.4 Chapter conclusions

The finite state models relying on input/output sequences of service specification augmented with appropriate parameters can increase the accuracy of the QoS evaluation in some cases, compared to linear combinations and workflow-based aggregation functions. We have also discussed how the quality of a composite web service can be evaluated by the use of such models and how the selection of a QoS-aware component service can be performed. Also, time-related quality parameters are remaining one of the crucial for evaluating the service quality and in many services has an important influence on end-user satisfaction as well.

Therefore, in the following chapters we investigate the finite state model augmented with timed parameters in more details.

Chapter 3

Finite State Machines with Timeouts as a Formal Model for Web Services

3.1 Introduction

The finite state models proved their usefulness for various purposes while dealing with discrete event systems - including web services, especially while considering composite services: often [17–20] the service composition functional specification is treated as a conversation protocol, described with variants of finite automata and finite state machines, so that formal algorithms for its optimization and verification can be applied.

Similar to [21–23], to integrate service quality parameters into a functional model, we augment classical finite state model with additional functions. In the thesis, we integrate into the functional model one quality parameter - response time, for several reasons: first, it is one of the most popular quality parameter to consider while evaluating service quality and SLA negotiations; second, for wide range of services it also has a great impact on user satisfaction; third, it is often useful for detecting some safety issues in communication between several components; forth, response time is one of the most representative cumulative (or, additive) parameter, i.e., the aggregation of such parameters in complex processes is based on sum operator (for example, in aggregation functions in Table 2.1).

Therefore, in the thesis we consider the specification and components behavior being represented with Finite State Machines with Timeouts (TFSM): classical FSMs augmented with timed functions for input and output symbols. In this Chapter, we investigate essential properties and relations for the TFSM model, starting from the general definition. Then, we propose a nondeterministic extension to model lack of control and observability while dealing with third-party components. For verification and testing purposes, the conformance relations are defined over TFSMs, and we show that it is sufficient to consider TFSM behavior only at integer-valued time instances. The latter allows to establish the correspondence between TFSMs and classical Finite Automata (FA), opening the way for the efficient adaptation of well-developed formal techniques. The established correspondence holds due to the cumulative nature of time parameter, and in the future work might be investigated for other than time quality parameters and metrics.

3.2 TFSM definitions and notations

3.2.1 General definition

Definition 3.1. A *Finite State Machine with Timeouts* (TFSM) is a 7-tuple $\mathcal{S} = \langle S, I, O, \lambda_{\mathcal{S}}, s_0, \Delta_{\mathcal{S}}, \sigma_{\mathcal{S}} \rangle$, where:

- the 5-tuple $\langle S, I, O, \lambda_{\mathcal{S}}, s_0 \rangle$ is an (underlaying) FSM:
  - S is the finite non-empty set of states with the designated initial state s_0;
  - I and O are the input and the output alphabets, respectively;
  - $\lambda_{\mathcal{S}} \subseteq S \times I \times O \times S$ is the transition relation;
- $\Delta_{\mathcal{S}} : S \to S \times (\mathbb{N} \cup \{\infty\})$ is a *timeout function*, and
- $\sigma_{\mathcal{S}} : \lambda_{\mathcal{S}} \to Time$ is an output *delay function*.

The timeout function $\Delta_{\mathcal{S}}(s) = (s_T, T)$ prescribes for each state $s \in S$ the maximal time $T \in (\mathbb{N} \cup \{\infty\})$ (timeout) of the idle waiting at the state s for an input to be applied; and the next state $s_T \in S$ which the machine moves to if no input has been applied

before the timeout expires. If $\Delta_{\mathcal{S}}(s) = (s_T, \infty)$, then by definition, $s_T = s$, i.e., the machine can stay waiting for an input at state s infinitely long.

The output delay $\sigma_{\mathcal{S}} : \lambda_{\mathcal{S}} \to Time$ function defines for each transition $\langle s_1, i, s_2, o\rangle \in \lambda_{\mathcal{S}}$ the set of timed intervals $\{[l; r) | l < r \wedge l, r \in (\mathbb{N} \cup \{0, \infty\})\}$ within which the machine can process the applied input, execute the transition and produce the output [64–67]. △

Each TFSM has an internal time variable – *timer* – which is reset to 0 when either of the following occurs:

- TFSM is reset (i.e. TFSM current state is set to the initial state);
- TFSM receives an input;
- TFSM produces an output;
- TFSM state changes (due to input-output transition or timeout).

In other words, the value of TFSM timer shows how much time has passed since the TFSM reached its current state or received the input that is currently processed.

Definition 3.2. A TFSM is *complete* if the underlying FSM is complete; the latter means that for each state $s \in S$ the behavior of the machine is defined for any input $i \in I$, i.e., for each state $s \in S$ and input $i \in I$ there is at least one pair $(o, s') \in O \times S$ such that $\langle s, i, o, s'\rangle$ is transition of the underlying FSM; otherwise the TFSM is called *partial*.

For a given state $s \in S$, we denote $inputs_S(s)$ the set of inputs that are defined in the state s, i.e.,

$$inputs_S(s) = \{i \in I | \exists (o, s') \in O \times S : \langle s, i, o, s'\rangle \in \lambda_{\mathcal{S}}\}.$$

Definition 3.3. A TFSM is *functionally-deterministic* (*f-deterministic*) if the underlying FSM is deterministic, i.e., for each state $s \in S$ and input $i \in I$ there is at most one pair $(o, s') \in O \times S$ such that $\langle s, i, o, s'\rangle$ is transition of the underlying FSM.

A TFSM is *time-deterministic* (*t-deterministic*) if for each transition $\langle s, i, o, s'\rangle \in \lambda_{\mathcal{S}}$ the output delay function is a single interval, i.e., for each $\langle s, i, o, s'\rangle \in \lambda_{\mathcal{S}}$ it holds that $\sigma_{\mathcal{S}}(\langle s, i, o, s'\rangle) = \{[k; k + 1)\}, k \in \mathbb{N} \cup 0$.

A TFSM is *deterministic* if it is both f- and t-deterministic, otherwise, the TFSM is called *nondeterministic*. △

Example 3.1. *The* Vacation Planner service*, described as weighted automaton in Example 2.3, also can be represented by TFSM model in Fig. 3.1, where weights are wrapped as output delays values and requests and responses are paired. The resulting TFSM is partial and functionally-deterministic.*

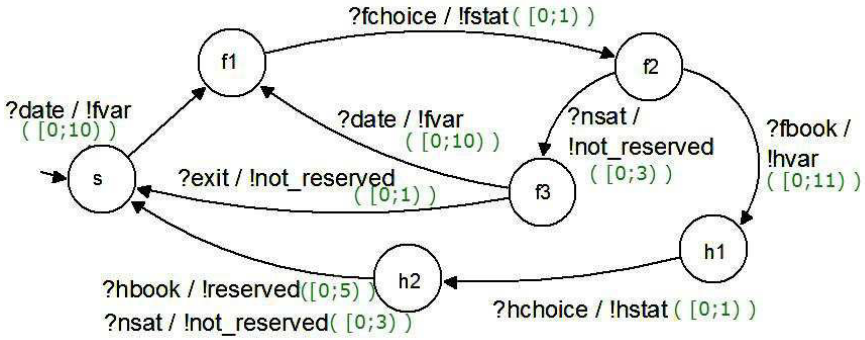

FIGURE 3.1: TFSM description of Vacation Planner service

3.2.2 TFSM behavior description

The behavior of the TFSM is characterized by the set of (timed) traces it accepts. Further we denote the set of non-negative real as $\mathbb{R}^+$.

Definition 3.4. A *timed input* is a pair $\langle i, t\rangle \in I \times \mathbb{R}^+$ which indicates that an input i is applied when TFSM time variable (timer) has value t, i.e. at the time instance t after the previous output was observed or the current state was reached.

A *timed output* is a pair $\langle o, t\rangle \in O \times \mathbb{R}^+$ which indicates that an output o is produced when the timer has value t, i.e. exactly at the moment t after an input was applied.

A sequence $\langle i_1, t_1\rangle \dots \langle i_m, t_m\rangle$ of timed inputs is a *timed input sequence*, while a sequence $\langle o_1, k_1\rangle \dots \langle o_m, k_m\rangle$ of timed outputs is a timed output sequence. △

Given the state s and input $i \in inputs_S(s)$, denote $outputs_S(s, i) \subseteq O \times \{\sigma_S\}$ the set of all output symbols paired with their possible delays, i.e.,

$$outputs_S(s, i) = \{\langle o, \sigma_S(tr)\rangle \mid \forall o \in O\, \exists s' \in S : tr = \langle s, i, o, s'\rangle \in \lambda_S\}.$$

Given the state s and input $i \in inputs_S(s)$, denote $next\_states_S(s,i) \subseteq S$ the set of *successors* of the state s under input i, i.e.,

$$next\_states_S(s,i) = \{s' \mid \exists o \in O : \langle s,i,o,s'\rangle \in \lambda_S\}.$$

In order to extend the transition relation on timed inputs and outputs, we define the following functions:

- $time_S : S \times \mathbb{R}^+ \to S$ – for a given state s and time value t, function $time_S$ computes the state of the TFSM t time units after reaching the state s according to the timeout function;
- $clock_S : S \times \mathbb{R}^+ \to \mathbb{R}^+$ – for a given state s and time value t, function $clocks_S$ computes the value of the timer t time units after reaching the state s.

To calculate the value of $time_S(s,t)$, we calculate the chain of timeout transitions $\Delta_S(s) = (s_1,T_1)$, $\Delta_S(s_1) = (s_2,T_2)$, ..., $\Delta_S(s_{p-1}) = (s_p,T_p)$ such that $T_1+T_2+\ldots+T_{p-1} < t$, but $T_1+T_2+\ldots+T_p \geq t$. In this case, it holds $time_S(s,t) = s_p$ and the value of time variable is $clock_S(s,t) = t-(T_1+T_2+\ldots+T_{p-1})$, since the last reset of the timer is performed when reaching the state s_p.

If $\Delta_S(s) = (s,\infty)$, then $time_S(s,t) = s$ and for any value of t, it is set $clock_S(s,t) = 0$.

By definition, the following properties hold: for $\Delta_S(s) = (s',T)$

$$time_S(s,t+T) = time_S(time_S(s,T),t),$$

$$clock_S(s,t+T) = clock_S(time_S(s,T),t).$$

In order to calculate the output that TFSM produces in response to a timed input $\langle i,t\rangle$ applied in state s, we first compute to which state s_p TFSM moves according to the timeout function after waiting t timed units, i.e., the input i is applied when TFSM is in the state $s_p = time_S(s,t)$.

Therefore, for each timed input $\langle i,t\rangle$ the transition relation λ_S is extended with transition $\langle s,\langle i,t\rangle,o,s'\rangle$, i.e., $\langle s,\langle i,t\rangle,o,s'\rangle \in \lambda^t_S$, and output delay function is defined as $\sigma_S\langle s,\langle i,t\rangle,o,s'\rangle = \sigma_S(\langle time_S(s,t),i,o,s'\rangle)$.

Receiving the input i in s_p, the TFSM can produce an output o if there is a transition $\langle s_p, i, o, s' \rangle \in \lambda_{\mathcal{S}}$ in response after any time instance $t' \in \sigma_{\mathcal{S}}(\langle s_p, i, o, s' \rangle)$ (i.e., t' being within some interval from $\sigma_{\mathcal{S}}(\langle s_p, i, o, s' \rangle)$).

The corresponding timed output $\langle o, t' \rangle$ is a possible response of the TFSM to the timed input $\langle i, t \rangle$ applied at the state s.

Consider the sequence $\alpha = \langle i_1, t_1 \rangle \langle i_2, t_2 \rangle \dots \langle i_n, t_n \rangle$. Timed input sequence α is called *acceptable* by TFSM $\mathcal{S}$ in state s if there exist a timed output sequence $\beta = \langle o_1, k_1 \rangle \langle o_2, k_2 \rangle \dots \langle o_n, k_n \rangle$ and a chain of states $s_1, s_2, \dots, s_n$ such that $\lambda^t_{\mathcal{S}}$ contains transitions $\langle s, \langle i_1, t_1 \rangle, o_1, s_1 \rangle$, $\langle s_1, \langle i_2, t_2 \rangle, o_2, s_2 \rangle$, $\dots$ $\langle s_{n-1}, \langle i_n, t_n \rangle, o_n, s_n \rangle$ and output delays are $k_1 \in \sigma_{\mathcal{S}}(\langle s, \langle i_1, t_1 \rangle, o_1, s_1 \rangle)$, $k_2 \in \sigma_{\mathcal{S}}(\langle s_1, \langle i_2, t_2 \rangle, o_2, s_2 \rangle)$, $\dots$ $k_n \in \sigma_{\mathcal{S}}(\langle s_{n-1}, \langle i_n, t_n \rangle, o_n, s_n \rangle)$.

Then, α/β is called the *timed trace* of the TFSM $\mathcal{S}$ in state s, and the transition relation is extended with the transition $\langle s, \alpha, \beta, s_n \rangle \in \lambda^{\alpha}_{\mathcal{S}}$.

Denote the set of all traces of $\mathcal{S}$ in state s as $trace^{\mathbb{R}}_S(s)$ and the set of all timed output sequences β, such that α/β is a timed trace, as $out^{\mathbb{R}}_S(s, \alpha)$.

For notation simplicity, we further omit superscripts t and α in extended transition relations and use $\lambda_{\mathcal{S}}$ everywhere. Where appropriate, we also use the following notation for transitions and timeout transitions: $\langle s, i, o, s' \rangle \in \lambda_{\mathcal{S}}$ with output delays $K = \sigma_{\mathcal{S}}(\langle s, i, o, s' \rangle)$ as $s \xrightarrow{i/o(K)} s'$; and timeout $\Delta_{\mathcal{S}}(s) = (q, T)$ as timeout transition $s \xrightarrow{T} q$.

Further, we also refer to the set of all traces of TFSM $\mathcal{S}$ in the initial state $trace^{\mathbb{R}}_S(s_0)$ as *the behavior of TFSM* $\mathcal{S}$.

Remark 3.5. For the sake of consistency, we further consider only TFSMs under the following assumptions:

- Since we condiser only initilized TFSMs, we also assume TFSMs to be *initially connected*, i.e., that each state of the TFSM is reachable form the initial state. In other words, that for each $s \in S$ there exists $\alpha/\beta \in trace^{\mathbb{R}}_S(s_0)$ so that $\langle s_0, \alpha, \beta, s \rangle \in \lambda_{\mathcal{S}}$.
- In case of partial nondeterministic TFSMs, we further assume that all traces are *harmonized* [68], meaning that acceptance of input sequences in states does

not depend on outputs. In other words, we require for any $s, s_1, s_2 \in S$ that if $\langle s, \alpha, \beta_1, s_1 \rangle \in \lambda_{\mathcal{S}}$ and $\langle s, \alpha, \beta_2, s_2 \rangle \in \lambda_{\mathcal{S}}$ then $inputs_S(s_1) = inputs_S(s_2)$. △

In terms of traces, we can redefine notions of determinism and completeness as follows. The TFSM $\mathcal{S}$ is complete if any timed input sequence α is acceptable in the initial state of $\mathcal{S}$, otherwise the TFSM $\mathcal{S}$ is partial [1]. The TFSM $\mathcal{S}$ is deterministic if for any timed input sequence α there exists at most one timed output sequence β so that α/β is a timed trace of the TFSM $\mathcal{S}$, otherwise the TFSM $\mathcal{S}$ is nondeterministic.

The set of all acceptable timed input sequences in the state s is further denoted as

$$in_S^{\mathbb{R}}(s) = \{\alpha \in (I \times \mathbb{R}^+)^* \mid \exists \beta \in (O \times \mathbb{R}^+)^* : \alpha/\beta \in trace_S^{\mathbb{R}}(s)\}.$$

3.3 Conformance relations for TFSMs

In order to design and analyze different interactive systems formal relations between two systems have to be established to compare their behaviors. For FSMs, such relations are well defined; and this is another reason why we use this model for service design and analysis. We modify these relations for FSMs with timeouts.

Definition 3.6. TFSMs $\mathcal{S}$ and $\mathcal{P}$ over the same input and output alphabets are *equivalent*, written $S \cong P$, if the sets of their traces coincide, i.e., it holds that $trace_S^{\mathbb{R}}(s_0) = trace_P^{\mathbb{R}}(p_0)$; otherwise, TFSMs $\mathcal{S}$ and $\mathcal{P}$ are *distinguishable*. In other words, for the equivalent TFSMs, the sets of defined timed input sequences coincide (they have the same specification domain) and the equivalent TFSMs produce the same output sequences in response to each defined timed input sequence. Formally, $trace_S^{\mathbb{R}}(s_0) = trace_P^{\mathbb{R}}(p_0)$ iff $in_S^{\mathbb{R}}(s_0) = in_P^{\mathbb{R}}(p_0)$ and for all $\alpha \in in_S^{\mathbb{R}}(s_0)$ it holds that $out_S^{\mathbb{R}}(s_0, \alpha) = out_P^{\mathbb{R}}(p_0, \alpha)$. △

Sometimes, it is said that the equivalent TFSMs have the same behavior.

Definition 3.7. The TFSM $\mathcal{S}$ is called a *reduction* of the TFSM $\mathcal{P}$, denote $\mathcal{S} \leqslant \mathcal{P}$, if $trace_S^{\mathbb{R}}(s_0) \subseteq trace_P^{\mathbb{R}}(p_0)$, i.e., if sets of defined timed input sequences of $\mathcal{S}$ is a subset

[1] It is sufficient in this definition that the completeness requirement is set for the initial state only, because we consider all traces to be harmonized and the TFSM being initially connected. Otherwise, the requirement should be set for each state of the TFSM, since some input sequences may lead to both complete and partial states with different output responses.

of that of $\mathcal{P}$ and for each defined timed input sequence, the output response of TFSM $\mathcal{S}$ to this sequence is in the set of output responses of $\mathcal{P}$ to this sequence. Formally, $in_S^{\mathbb{R}}(s_0) \subseteq in_P^{\mathbb{R}}(p_0)$ and for all $\alpha \in in_S^{\mathbb{R}}(s_0)$ it holds $out_S^{\mathbb{R}}(s_0, \alpha) \subseteq out_P^{\mathbb{R}}(p_0, \alpha)$. △

The equivalence and reduction relations are defined regardless of whether compared TFSMs are complete and deterministic or partial and nondeterministic. For partial TFSMs the equivalence means that behaviors of both compared TFSMs should be defined on the same sets of input sequences.

As service specifications are often partial and nondeterministic, special corresponding relations between partial FSMs should be adapted to TFSMs, namely, *quasi-equivalence* and *quasi-reduction* [68, 69].

Definition 3.8. The TFSM $\mathcal{S}$ is called *quasi-equivalent* to the TFSM $\mathcal{P}$, $\mathcal{S} \sqsupseteq \mathcal{P}$, if $in_S^{\mathbb{R}}(s_0) \supseteq in_P^{\mathbb{R}}(p_0)$ and for all $\alpha \in in_P^{\mathbb{R}}(p_0)$ it holds $out_S^{\mathbb{R}}(s_0, \alpha) = out_P^{\mathbb{R}}(p_0, \alpha)$, i.e. $\mathcal{S}$ and $\mathcal{P}$ have the same output responses to all the input sequences that are accepted by $\mathcal{P}$. △

Definition 3.9. The TFSM $\mathcal{S}$ is called a *quasi-reduction* of the TFSM $\mathcal{P}$, $S \lesssim P$, if $in_S^{\mathbb{R}}(s_0) \supseteq in_P^{\mathbb{R}}(p_0)$ and for all $\alpha \in in_P^{\mathbb{R}}(p_0)$ it holds $out_S^{\mathbb{R}}(s_0, \alpha) \subseteq out_P^{\mathbb{R}}(p_0, \alpha)$, i.e. for all input sequences accepted by the TFSM $\mathcal{P}$, the TFSM $\mathcal{S}$ can produces some of output responses produced by $\mathcal{P}$. △

Proposition 3.10. *Given TFSMs $\mathcal{S}$ and $\mathcal{P}$ with harmonized traces, the following conformance relations properties hold.*

1. *$\mathcal{S}$ and $\mathcal{P}$ are equivalent if and only if $\mathcal{S}$ is a reduction of $\mathcal{P}$ and $\mathcal{P}$ is a reduction of $\mathcal{S}$: $\mathcal{S} \cong \mathcal{P}$ if and only if $\mathcal{S} \leq \mathcal{P}$ and $\mathcal{P} \leq \mathcal{S}$.*
2. *$\mathcal{S}$ and $\mathcal{P}$ are equivalent if and only if $\mathcal{S}$ is a quasi-equivalent to $\mathcal{P}$ and $\mathcal{P}$ is quasi-equivalent to $\mathcal{S}$: $\mathcal{S} \cong \mathcal{P}$ if and only if $\mathcal{S} \sqsupseteq \mathcal{P}$ and $\mathcal{P} \sqsupseteq \mathcal{S}$.*
3. *$\mathcal{S}$ and $\mathcal{P}$ are equivalent if and only if $\mathcal{S}$ is a quasi-reduction of $\mathcal{P}$ and $\mathcal{P}$ is a quasi-reduction of $\mathcal{S}$: $\mathcal{S} \cong \mathcal{P}$ if and only if $\mathcal{S} \lesssim \mathcal{P}$ and $\mathcal{P} \lesssim \mathcal{S}$.*
4. *If $\mathcal{S}$ is a quasi-reduction of $\mathcal{P}$ and $\mathcal{P}$ is a reduction of $\mathcal{S}$, then $\mathcal{S}$ is quasi-equivalent to $\mathcal{P}$: if $\mathcal{S} \lesssim \mathcal{P}$ and $\mathcal{P} \leq \mathcal{S}$ then $\mathcal{S} \sqsupseteq \mathcal{P}$.*

Proof. 1. $\mathcal{S} \cong \mathcal{P}$ if and only if $\mathcal{S} \leq \mathcal{P}$ and $\mathcal{P} \leq \mathcal{S}$

The property holds by definition, since $trace_S^{\mathbb{R}}(s_0) = trace_P^{\mathbb{R}}(p_0)$ if and only if $trace_S^{\mathbb{R}}(s_0) \subseteq trace_P^{\mathbb{R}}(p_0)$ and $trace_P^{\mathbb{R}}(p_0) \subseteq trace_S^{\mathbb{R}}(s_0)$.

2. $\mathcal{S} \cong \mathcal{P}$ if and only if $\mathcal{S} \sqsupseteq \mathcal{P}$ and $\mathcal{P} \sqsupseteq \mathcal{S}$

   (a) $\mathcal{S} \cong \mathcal{P}$ holds, by definition, if and only if $trace_S^{\mathbb{R}}(s_0) = trace_P^{\mathbb{R}}(p_0)$, hence, if and only if $in_S^{\mathbb{R}}(s_0) = in_P^{\mathbb{R}}(p_0)$ and for all $\alpha \in in_S^{\mathbb{R}}(s_0)$ it holds $out_S^{\mathbb{R}}(s_0, \alpha) = out_P^{\mathbb{R}}(p_0, \alpha)$.

   (b) $in_S^{\mathbb{R}}(s_0) = in_P^{\mathbb{R}}(p_0)$ if and only if $in_S^{\mathbb{R}}(s_0) \subseteq in_P^{\mathbb{R}}(p_0)$ and $in_P^{\mathbb{R}}(p_0) \subseteq in_S^{\mathbb{R}}(s_0)$.

   By definition of quasi-equivalence, if $in_S^{\mathbb{R}}(s_0) \subseteq in_P^{\mathbb{R}}(p_0)$ and for all $\alpha \in in_S^{\mathbb{R}}(s_0)$ it holds $out_S^{\mathbb{R}}(s_0, \alpha) = out_P^{\mathbb{R}}(p_0, \alpha)$ means $\mathcal{S} \sqsupseteq \mathcal{P}$.

3. $\mathcal{S} \cong \mathcal{P}$ if and only if $\mathcal{S} \lesssim \mathcal{P}$ and $\mathcal{P} \lesssim \mathcal{S}$

   (a) $\mathcal{S} \cong \mathcal{P}$ holds, by definition, if and only if $trace_S^{\mathbb{R}}(s_0) = trace_P^{\mathbb{R}}(p_0)$, hence, if and only if $in_S^{\mathbb{R}}(s_0) = in_P^{\mathbb{R}}(p_0)$ and for all $\alpha \in in_S^{\mathbb{R}}(s_0)$ it holds $out_S^{\mathbb{R}}(s_0, \alpha) = out_P^{\mathbb{R}}(p_0, \alpha)$.

   (b) $in_S^{\mathbb{R}}(s_0) = in_P^{\mathbb{R}}(p_0)$ if and only if $in_S^{\mathbb{R}}(s_0) \subseteq in_P^{\mathbb{R}}(p_0)$ and $in_S^{\mathbb{R}}(s_0) \supseteq in_P^{\mathbb{R}}(p_0)$

   (c) and for all $\alpha \in in_S^{\mathbb{R}}(s_0) = in_P^{\mathbb{R}}(p_0)$ it holds $out_S^{\mathbb{R}}(s_0, \alpha) = out_P^{\mathbb{R}}(p_0, \alpha)$ if and only if $out_S^{\mathbb{R}}(s_0, \alpha) \subseteq out_P^{\mathbb{R}}(p_0, \alpha)$ for all $\alpha \in in_P^{\mathbb{R}}(p_0)$ and $out_S^{\mathbb{R}}(s_0, \alpha) \supseteq out_P^{\mathbb{R}}(p_0, \alpha)$ for all $\alpha \in in_S^{\mathbb{R}}(s_0)$.

   The latter matches the definition of quasi-reduction.

4. if $\mathcal{S} \lesssim \mathcal{P}$ and $\mathcal{P} \leq \mathcal{S}$ then $\mathcal{S} \sqsupseteq \mathcal{P}$

   If $\mathcal{S} \lesssim \mathcal{P}$ then it holds $in_S^{\mathbb{R}}(s_0) \supseteq in_P^{\mathbb{R}}(p_0)$ and $out_S^{\mathbb{R}}(s_0, \alpha) \subseteq out_P^{\mathbb{R}}(p_0, \alpha)$ for all $\alpha \in in_P^{\mathbb{R}}(p_0)$.

   If $\mathcal{P} \leq \mathcal{S}$ then it holds $trace_S^{\mathbb{R}}(s_0) \supseteq trace_P^{\mathbb{R}}(p_0)$, therefore, $in_P^{\mathbb{R}}(p_0) \subseteq in_S^{\mathbb{R}}(s_0)$ and $out_P^{\mathbb{R}}(p_0, \alpha) \subseteq out_S^{\mathbb{R}}(s_0, \alpha)$ for all $\alpha \in in_P^{\mathbb{R}}(p_0)$.

   Hence, $in_S^{\mathbb{R}}(s_0) \supseteq in_P^{\mathbb{R}}(p_0)$ and $out_S^{\mathbb{R}}(s_0, \alpha) = out_P^{\mathbb{R}}(p_0, \alpha)$ for all $\alpha \in in_P^{\mathbb{R}}(p_0)$, which matches the definition of quasi-equivalence.

□

3.4 From real to integer values of time variable

The problem now is that all these relations are defined over sets of timed sequences for real time instances and, hence, time variable has infinite number of values within any interval. But since all the timeouts in the TFSMs and boundaries of intervals for output delays are integers and time variable is reset after each transition, we can restrict the sets of traces to integer time instances preserving the above conformance relations between TFSMs.

3.4.1 Integer-valued conformance relations

Given TFSM $\mathcal{S}$, the following formula can be established.

First, for any state s it holds that $time_S(s, n+\delta) = time_S(s, n)$ for all $n \in \mathbb{N} \cup \{0\}$ and $\delta \in [0, 1)$:

1. if $\Delta_{\mathcal{S}}(s) = (s', T)$ and $n < T$ or $\Delta_{\mathcal{S}}(s) = (s, \infty)$ then $n + \delta < T$ and $time_S(s, n + \delta) = time_S(s, n) = s$;
2. if $\Delta_{\mathcal{S}}(s) = (s', T)$ and $n = T$ then $time_S(s, n + \delta) = time_S(s', \delta) = s'$ since $\delta < 1$;
3. if $\Delta_{\mathcal{S}}(s) = (s', T)$ and $n > T$ then $time_S(s, n + \delta) = time_S(s', n - T + \delta)$ which iteratively is reduced to the clauses $n \leq T$.

Therefore, $\langle s, \langle i, n + \delta\rangle, o, s_n\rangle \in \lambda_{\mathcal{S}}$ if and only if $\langle s, \langle i, n\rangle, o, s_n\rangle \in \lambda_{\mathcal{S}}$.

Consider a timed input sequence $\alpha = \langle i_1, t_1\rangle\langle i_2, t_2\rangle \dots \langle i_k, t_k\rangle$ where $t_j = n_j + \delta_j$, $\delta_j \in [0, 1)$ for all $1 \leq j \leq k$. We denote $\alpha^{int} = \langle i_1, n_1\rangle\langle i_2, n_2\rangle \dots \langle i_k, n_k\rangle$.

Proposition 3.11. *For any state s it holds that $\alpha \in in_S^{\mathbb{R}}(s)$ if and only if $\alpha^{int} \in in_S^{\mathbb{R}}(s)$.*

Proof. Consider two sequences $\alpha = (i, n + \delta)\alpha'$ and $\alpha_1 = (i, n)\alpha'$.

It holds that $\alpha \in in_S^{\mathbb{R}}(s)$ if and only if $i \in inputs_S(time_S(s, n + \delta))$, i.e., there exist s_1 and o such that $\langle time_S(s, n+\delta), i, o, s_1\rangle \in \lambda_{\mathcal{S}}$, and $\alpha' \in in_S^{\mathbb{R}}(s_1)$. Since $time_S(s, n+\delta) = time_S(s, n)$, it holds that $i \in inputs_S(time_S(s, n))$ and $\alpha' \in in_S^{\mathbb{R}}(s_1)$ implies $\alpha_1 \in in_S^{\mathbb{R}}(s)$. The proposition then can be proven using the induction on the length of sequence α'. □

Proposition 3.12. *For any state s and any input sequence α, it holds that $out_S^{\mathbb{R}}(s, \alpha) = out_S^{\mathbb{R}}(s, \alpha^{int})$.*

Proof. The proof is a corollary (by induction) of the property that for all $\delta \in [0, 1)$ it holds $\langle s, \langle i, n + \delta\rangle, o, s_n\rangle \in \lambda_{\mathcal{S}}$ if and only if $\langle s, \langle i, n\rangle, o, s_n\rangle \in \lambda_{\mathcal{S}}$. □

On the other hand, we consider that the global timer which measures time intervals between applying an input and observing an output is most likely discrete, then if an output o is produced at some time instance $n+\delta$, $n \in \mathbb{N} \cup \{0\}$ and $\delta \in [0, 1)$, we observe the timer value n and denote this timed output as (o, n). Therefore, timed output (o, n) is interpreted as an output o produced at any time instance between $[n, n + 1)$ after a corresponding input was applied, and two timed outputs (o, n) and $(o, n+\delta)$ are considered equivalent for all $n \in \mathbb{N} \cup \{0\}$ and $\delta \in [0, 1)$ and treated as the same output.

Denote $in_S(s) = \{\alpha^{int} \mid \alpha \in in_S^{\mathbb{R}}(s)\}$ the set of integer-valued timed input sequences defined in the state s, the $out_S(s, \alpha) = \{\beta^{int} \mid \beta \in out_S^{\mathbb{R}}(s, \alpha)\}$ the set of integer-valued timed output sequences in response to the sequence α in state s, while $trace_S(s) = \{\alpha^{int}/\beta^{int} \mid \alpha/\beta \in trace_S^{\mathbb{R}}(s)\}$ denotes the set of all integer-valued timed traces in state s.

Proposition 3.13. *Given TFSMs $\mathcal{S}$ and $\mathcal{P}$, the following statements holds:*

1. *$\mathcal{S} \cong \mathcal{P}$ iff $trace_S(s_0) = trace_P(p_0)$;*
2. *$\mathcal{S} \leqslant \mathcal{P}$ iff $trace_S(s_0) \subseteq trace_P(p_0)$;*
3. *$\mathcal{S} \sqsupseteq \mathcal{P}$ iff $in_S(s_0) \supseteq in_P(p_0)$ and for all $\alpha \in in_P(p_0)$ it holds $out_S(s_0, \alpha) = out_P(p_0, \alpha)$;*
4. *$\mathcal{S} \lesssim \mathcal{P}$ iff $in_S(s_0) \supseteq in_P(p_0)$ and for all $\alpha \in in_P(p_0)$ it holds $out_S(s_0, \alpha) \subseteq out_P(p_0, \alpha)$;*

Proof. Directly follows from Propositions 3.11 and 3.12. □

According to Proposition 3.13, when talking about conformance relations it is sufficient to consider timed traces where time instances are integers.

Correspondingly, we can restrict the definition of the output delay function of TFSM from intervals to sets of integer values without breaking a considered conformance relation. Therefore, the following definition can be considered without loss of generality.

Definition 3.14. A *Finite State Machine with Timeouts* (TFSM) is a 7-tuple $\mathcal{S} = \langle S, I, O, \lambda_{\mathcal{S}}, s_0, \Delta_{\mathcal{S}}, \sigma_{\mathcal{S}} \rangle$, where:

- the 5-tuple $\langle S, I, O, \lambda_{\mathcal{S}}, s_0 \rangle$ is an FSM;
- $\Delta_{\mathcal{S}} : S \to S \times (\mathbb{N} \cup \{\infty\})$ is a *timeout function*, and
- $\sigma_{\mathcal{S}} : \lambda_{\mathcal{S}} \to 2^{\mathbb{N} \cup \{0\}}/\emptyset$ is an *output delay function* that defines the non-empty (possibly, infinite) set of allowed delays for the machine. △

Moreover, in this thesis, we assume that the set of values of $\sigma_{\mathcal{S}}$ for each transition can be represented as a set of values of linear functions $\{b + kt \mid b, k \in \{0\} \cup \mathbb{N}\}$.

Since the timed behavior aspects of TFSM can be adequately modeled at integer time instances, it is possible to represent waiting for one time unit using dedicated discrete action `1` and characterize the behavior of the TFSM with a regular language of a corresponding finite automaton.

3.4.2 Corresponding Finite Automata

The behavior of a TFSM can be described by a corresponding regular language and, similar to classical FSMs, the composition of TFSMs can be derived based on operations over corresponding finite automata (FA). Therefore, in this section, we first remind necessary operations over finite automata and then define the correspondence between TFSMs and FA.

A *Finite Automaton* (FA) is a 5-tuple $\mathcal{P} = (P, A, \delta_P, p_0, Q)$, where P is finite non-empty set of states with the designated initial state p_0 and the set $Q \subseteq P$ of final (accepting) states, A is a finite alphabet of actions, $\delta_P \subseteq P \times (A \times \epsilon) \times P$ is the transition relation.

Further, we also refer to a finite automaton simply as an automaton where it does not cause any ambiguity.

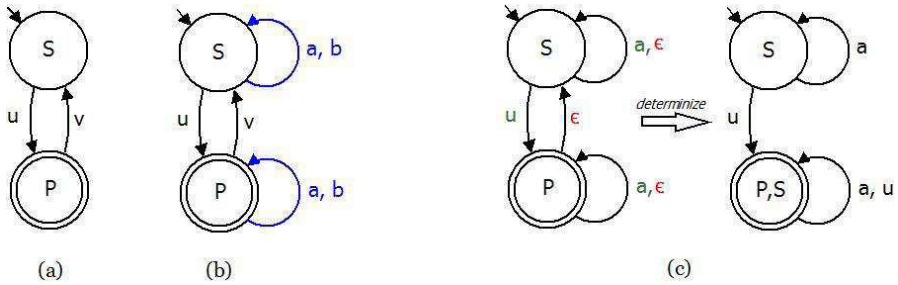

FIGURE 3.2: Automaton P (a), expansion of P to alphabet $\{a, b\}$ (b), and restriction of $P_{\Uparrow\{a,b,c\}}$ to alphabet $\{a, u\}$ (c)

For each state $p \in P$ denote the set of actions under which an outgoing transitions are defined as $actions_P(p) = a \mid \exists p' \in P\langle p, a, p'\rangle \in \delta_P$.

By induction, the transition relation is extended to sequences of A^*. The automaton *accepts* (recognizes) a string $\alpha \in A^*$ if there exists a state $q \in Q$ such that $(p_0, \alpha, q) \in \delta_P$. The set of all strings accepted by the automaton is the language of the automaton and is denoted $L(\mathcal{P})$. Automata $\mathcal{P}$ and $\mathcal{S}$ are *equivalent* if they accept the same language, i.e., $L(\mathcal{P}) = L(\mathcal{S})$.

An FA $\mathcal{P}$ is *deterministic* if $\delta_P \subseteq P \times A \times P$ and for each pair $(p_1, a) \in P \times A$ there is at most one state $p_2 \in P$ such that $(p_1, a, p_2) \in \delta_P$, otherwise, the finite automaton is *nondeterministic*. It is known that for each nondeterministic automaton there exists an equivalent deterministic automaton which can be derived by the use of subset construction [70].

The *expansion* of an automaton $\mathcal{P} = (P, A, \delta_P, p_0, Q)$ to alphabet B (or the B-expansion) is the automaton $P_{\Uparrow B} = (P, A \cup B, \delta_P^B, p_0, Q)$ where $\delta_P^B = \delta_P \cup \{(p, b, p) | p \in P \wedge b \in B \setminus A\}$, $L(P_{\Uparrow B}) = L(P)_{\Uparrow B}$. [71] In other words, in order to get the B-expansion of the automaton $\mathcal{P}$ we add a self-loop at each state p labeled with every action from B that is not in the alphabet A. The *restriction* of an automaton $\mathcal{P}$ to alphabet $C \subseteq A$ is an the automaton $P_{\Downarrow C} = (P, C, \delta_P^C, p_0, Q)$ where for all $(p_1, a, p_2) \in \delta_P$, if $a \in C$ then $(p_1, a, p_2) \in \delta_P^C$, else if $a \in A \setminus C$ then $(p_1, \epsilon, p_2) \in \delta_P^C$.

In other words, to extend an automaton we add a loop at each state for each new symbol we extend to. To restrict an automaton we hide symbols we are not interested in by replacing them with an the empty move.

The expansion and restriction operations on automaton are shown in Figure 3.2.

By definition, the operators $\Uparrow$ and $\Downarrow$ possess the following features: $L(P_{\Uparrow B}) = L(P)_{\Uparrow B}$ and $L(P_{\Downarrow C}) = L(P)_{\Downarrow C}$[71].

Given two automata $\mathcal{P} = (P, A, \delta_P, p_0, Q_P)$ and $\mathcal{R} = (R, A, \delta_R, r_0, Q_R)$, the *intersection* $\mathcal{C} = \mathcal{P} \cap \mathcal{R}$ is the automaton $\mathcal{C} = (P \times R, A, \delta_C, \langle p_0, r_0\rangle, Q_P \times Q_R)$ so that for all $\langle p, r\rangle \in P \times R$ and all $a \in A$ it holds $\langle\langle p, r\rangle, a, \langle p', r'\rangle\rangle \in \delta_C$ if and only if $\langle p, a, p'\rangle \in \delta_P$ and $\langle r, a, r'\rangle \in \delta_R$. [70] Intersection operation can be considered as a special variant of more general product operation for deriving the common behavior of two automata. Due to automaton-language correspondence, it holds by definition that $L(\mathcal{P} \cap \mathcal{R}) = L(\mathcal{P}) \cap L(\mathcal{R})$. Further, referring to automata intersection, we also assume, that unreachable states are either deleted or not derived.

A finite automaton $Aut(\mathcal{S})$ *corresponds* to a given TFSM $\mathcal{S}$ if for each timed trace $\alpha\beta = \langle i_1, t_1\rangle.\langle o_1, k_1\rangle.\langle i_2, t_2\rangle.\langle o_2, k_2\rangle \ldots \langle i_n, t_n\rangle.\langle o_n, k_n\rangle$ of the TFSM $\mathcal{S}$ the automaton $Aut(\mathcal{S})$ has a trace $\mathtt{1}^{t_1} i_1 \mathtt{1}^{k_1} o_1 \ldots \mathtt{1}^{t_n} i_1 \mathtt{1}^{k_n} o_n$, where symbol $\mathtt{1}$ denotes the waiting for one time unit.

The general idea of deriving the corresponding automaton can be summarized by following rules.

1. Each transition $s \xrightarrow{i/o(k)} q$ of the TFSM is unfolded into a chain of transitions $s \xrightarrow{i} \langle s, i\rangle \underbrace{\xrightarrow{1} \ldots \xrightarrow{1}}_{k} \langle s, i, k\rangle \xrightarrow{o} q$, where all intermediate states (i.e. all states except s and q) are non-final.

2. For each infinite timeout the automaton has a loop under $\mathtt{1}$ in the corresponding state, i.e., if $\Delta_{\mathcal{S}}(s) = (s, \infty)$ then the automaton has a transition $s \xrightarrow{1} s$.

3. A timeout transition $s \xrightarrow{t} s'$ for $t < \infty$ is unfolded into a chain of transitions $s \underbrace{\xrightarrow{1} \langle s, 1\rangle \xrightarrow{1} \ldots \xrightarrow{1} \langle s, t-1\rangle \xrightarrow{1}}_{t} s'$, all intermediate states $\langle s, j\rangle$ being final and "copies" of s, i.e. has the same transitions under $i \in I$ as the state s.

In other words, we unfold each input/output transition of TFSM $\mathcal{S}$ and add several transitions labeled with the action $\mathtt{1}$ between an input and an output according to the output delay value. Timeout transitions (i.e., transitions according to the timeout function) are unfolded into a chain of transitions labeled with action $\mathtt{1}$ preceding a corresponding input, similar to that in [66] for corresponding FSM.

In more details, the process of deriving the corresponding automaton is shown in the Algorithm 3.2.

For t-nondeterministic TFSM, if output delay function is periodical for some transitions, it cannot be unfolded into a chain of transitions under $\mathtt{1}$ directly. Though, for any set of linear functions there exists a regular language (i.e., the language of some finite automaton) so that the set of length of its words coincides with the set of values of these linear functions.[2] In other words, any set of linear functions can be represented with some one-letter finite automaton. In our case, we unfold sets of linear functions to automata over one-letter alphabet $\{\mathtt{1}\}$, representing given set of output delays, in Algorithm 3.1. The algorithm, in general, returns non-deterministic finite automaton which can be determinized by any known procedure (e.g., see [70]) if needed. In the non-deterministic automaton, derived by Algorithm 3.1, the number of cycles is equal to the number of functions with non-zero period $k_j \neq 0$ in the set, with lengths k_j, and all cycles contain only one final state. After the determinization, the automaton will contain the single cycle of length $\mathrm{LCM}_j(k_j)$ with multiple final states.

Algorithm 3.1 Deriving the automaton representing the values of the set of linear functions

Input: The set of linear functions $\sigma = \{b_j + k_j t \mid (1 \leq j \leq n) \wedge (b_j, k_j \in \mathbb{N} \cup \{0\})\}$.
Output: The corresponding automaton $Aut(\sigma) = \langle P, \{\mathtt{1}\}, \delta_P, p_0, P_{fin} \rangle$, so that $\mathtt{1}^k \in L(Aut(\sigma))$ if and only if $k \in \sigma$.

1: **if** $0 \in \sigma$ **then** $p_0 \in P_{fin}$ **end if**
2: $b_{max} = \max_{1 \leq j \leq n}(b_j)$.
3: **for** all $1 \leq b \leq b_{max}$ **do** add state (p, b) to the set P
4: **end for**
5: **for** all $1 < b \leq b_{max}$ **do** add transition $\langle (p, b-1), \mathtt{1}, (p, b) \rangle$ to δ_P
6: **end for**
7: Add transition $\langle p_0, \mathtt{1}, (p, 1) \rangle$ to δ_P
8: **for** each $b_j \neq 0$ **do** set $(p, b_j) \in P_{fin}$
9: **end for**
10: **for** each $k_j \neq 0$ **do** a loop of length k_j from corresponding final state:
11: **for** all $1 < k < k_j$ **do**
12: Add state $(p, b_j)_{k-1}$ to P.
13: Add transition $\langle (p, b_j)_{k-1}, \mathtt{1}, (p, b_j)_k \rangle$ to δ_P
14: **end for**
15: Add transition $\langle (p, b_j), \mathtt{1}, (p, b_j)_1 \rangle$ to δ_P
16: Add transition $\langle (p, b_j)_{k_j - 1}, \mathtt{1}, (p, b_j) \rangle$ to δ_P
17: **end for**
18: Determinize derived $Aut(\sigma)$ if needed.

[2] The length sets of regular languages are known to be semi-linear (i.e., the finite unions of linear sets) [72], which can be shown as a particular case of the Parikh theorem for context-free grammars.

Given the TFSM $\mathcal{S}$, the corresponding automaton $Aut(\mathcal{S})$ is derived by Algorithm 3.2.

By construction, the following proposition holds.

Proposition 3.15. *Given a TFSM $\mathcal{S}$, an automaton $Aut(\mathcal{S})$ derived by Algorithm 3.2 accepts a trace $\alpha \in [1^{t_1} i_1 1^{k_1} o_1 \dots 1^{t_n} i_1 1^{k_n} o_n] 1^*$ if and only if the TFSM $\mathcal{S}$ has a trace $\langle i_1, t_1 \rangle . \langle o_1, k_1 \rangle \dots \langle i_n, t_n \rangle . \langle o_n, k_n \rangle$.*

Proof. The proposition holds by construction. Suffix 1^* after each trace means that TFSM does not stop functioning but continue waiting for a next input with corresponding changing of states according to the timeout function. □

Consider the TFSM $\mathcal{S} = \langle S, I, O, s_0, \lambda_{\mathcal{S}}, \Delta_{\mathcal{S}}, \sigma_{\mathcal{S}} \rangle$. Let T_{max} denote the maximal finite timeout, K_{max} - the maximal finite output delay without cycles, $n = |S|$, $m = |I|$, $k = |O|$, and l - the maximal number of output symbol variants for each input applied at a given state. In practice, the value of l is usually essentially less than $|O|$, and somehow can be used as a measure of functional nondeterminism of the TFSM. For f-deterministic TFSMs $l = 1$. Then, the corresponding automaton $Aut(\mathcal{S})$ has at most $n \cdot T_{max}$ final states (states of the TFSM plus their "copies "via timeout transitions unrolling). Each transition of the TFSM is unrolled into a chain of transitions starting from states of form $\langle s, i \rangle$ and there are at most $n \cdot m$ of those states. The last non-final states in such unrolled chains are states from which transitions under output symbols are defined to states from S, and hence there are at most $n \cdot k$ last non-final states of unrolled transition chains. The length of a chain of transitions and number of intermediate states it traverses are defined by the maximal output delay value. In case of periodic output delays, the maximal number of intermediate states is also defined by the maximal output delay value without going into cycles. The chain of transitions under 1 starting from states of form $\langle s, i \rangle$ can lead to at most l last non-final states due to f-nondeterminism. Therefore, there are at most $m \cdot l \cdot K_{max}$ such intermediate states.

Overall, the corresponding automaton has at most $n \cdot (T_{max} + m + k) + m \cdot l \cdot K_{max}$ states.

Taking into account that modern FA-processing tools (see, for example, BALM [71]) efficiently process finite automata with hundred thousands of states, and that this worst evaluation is practically unreachable (especially for partial TFSMs), the transformation

Algorithm 3.2 Deriving the corresponding automaton for given TFSM

Input: The TFSM $\mathcal{S} = \langle S,\ I,\ O,\ s_0,\ \lambda_{\mathcal{S}},\ \Delta_{\mathcal{S}}, \sigma_{\mathcal{S}}\rangle$

Output: The corresponding automaton $Aut(\mathcal{S}) = \langle Q \cup F, I \cup O \cup \{\mathtt{1}\}, \delta_S, s_0, Q\rangle$, with the set of final states $Q \subseteq S \cup (S \times \mathbb{N})$ and the set of non-final states $F \subseteq (S \times I) \cup (S \times I \times \mathbb{N})$.

Set $Q = S$, the initial state is s_0 .
Unfold all transitions
for each $tr = \langle s, i, o, s'\rangle \in \lambda_{\mathcal{S}}$ **do**
    Add state $\langle s, i\rangle$ to the set F.
    Add transition $\langle s, i, \langle s, i\rangle\rangle$ to δ_S.
    if $0 \in \sigma_{\mathcal{S}}(tr)$ **then** add transition $\langle\langle s, i\rangle, o, s'\rangle$ to δ_S **end if**
    if $\sigma_{\mathcal{S}}(tr) \neq \{0\}$ **then**
Unfold non-zero output delays
        if $|\sigma_{\mathcal{S}}(tr)| = n < \infty$ **then** $k_{max} = \max(k_j \in \sigma_{\mathcal{S}}(tr))$
Unfold deterministic delay into a simple chain of transitions
            for all $1 \leq k \leq k_{max}$ **do** add $\langle s, i, k\rangle$ to the set F
            end for
            for all $1 < k \leq k_{max}$ **do** add transition $\langle\langle s, i, k-1\rangle, \mathtt{1}, \langle s, i, k\rangle\rangle$ to δ_S
            end for
            Add transition $\langle\langle s, i\rangle, \mathtt{1}, \langle s, i, 1\rangle\rangle$ to δ_S
            for each $k_j \in \sigma_{\mathcal{S}}(tr)$ **do** add transition $\langle\langle s, i, k_j\rangle, o, s'\rangle$ to δ_S
            end for
        else if $\sigma_{\mathcal{S}}(tr) = \{b_1 + k_1 t; b_2 + k_2 t; \ldots b_n + k_n t\}$ **then**
Unfold non-deterministic delays into a finite automaton using source state of the TFSM transition as initial state for the automaton representing delays
            Derive an automaton $Aut(\sigma_{\mathcal{S}}(tr)) = \langle P, \{\mathtt{1}\}, \delta_P, p_0, P_{fin}\rangle$ by Algorithm 3.1
            Add P to the set of non-final states F
            Merge p_0 to $\langle s, i\rangle$
            Add all transitions from δ_P to δ_S
            for all $p \in P_{fin}$ **do** add $\langle p, o, s'\rangle$ to δ_S
            end for
        end if
    end if
end for
Unfold timeouts
for each $s \in S$ **do**
    if $\Delta_{\mathcal{S}}(s) = (s, \infty)$ **then** add transition $\langle s, \mathtt{1}, s\rangle$ to δ_S
    else if $\Delta_{\mathcal{S}}(s) = (s', 1)$ **then** add transition $\langle s, \mathtt{1}, s'\rangle$ to δ_S
    else if $\Delta_{\mathcal{S}}(s) = (s', T)$ with $1 < T < \infty$ **then**
        for each $1 \leq t < T$ **do**
            Add state $\langle s, t\rangle \in Q$
            for all $\langle s, i, \langle s, i\rangle\rangle \in \delta_S$ **do** add transition $\langle\langle s, t\rangle, i, \langle s, i\rangle\rangle$ to δ_S
            end for
        end for
        Add transition $\langle s, \mathtt{1}, \langle s, 1\rangle\rangle$ to δ_S
        for each $1 < t < T$ **do** add transition $\langle\langle s, t-1\rangle, \mathtt{1}, \langle s, t\rangle\rangle$ to δ_S
        end for
        Add transition $\langle\langle s, T-1\rangle, \mathtt{1}, s'\rangle$ to δ_S
    end if
end for

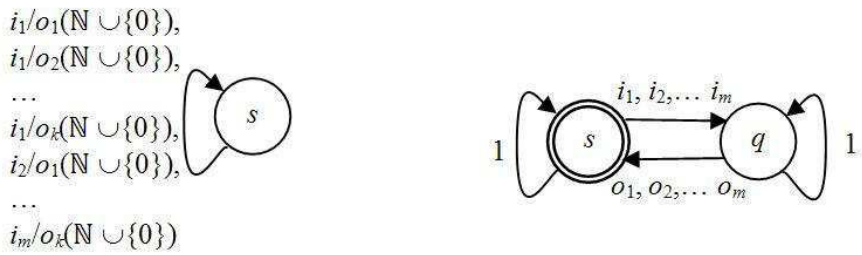

FIGURE 3.3: The maximal TFSM over alphabets $I = \{i_1 \ldots i_m\}$ and $O = \{o_1 \ldots o_k\}$ (left) and its corresponding automaton (right)

from TFSMs to FA and performing operations over TFSMs via operations over corresponding FA seems a practical approach. The bottle-neck of the approach lays in operations involving FA determinisation (which is known to have exponential complexity), and development of work-around solutions is an interesting perspective for future work.

The language $L(\mathcal{S})$ of the automaton $Aut(\mathcal{S})$ is further called the *time advanced language* of the TFSM $\mathcal{S}$, or further on just *language* of the TFSM $\mathcal{S}$ for short. Here we note that the language $L(\mathcal{S})$ indeed contains not only traces corresponding to timed traces of the TFSM $\mathcal{S}$, but also all their continuations with the suffixes of the set $\mathtt{1}^*$; the latter means, that after producing the tail output of the trace, the TFSM not necessary resets to the initial state or stops functioning but can continue to change its states according to the timeout function. Due to the correspondence of traces and languages and Proposition 3.15 the following proposition holds.

Proposition 3.16. *Given TFSMs $\mathcal{S}$ and $\mathcal{P}$, the TFSMs are equivalent (or $\mathcal{S}$ is reduction of $\mathcal{P}$), if and only if their languages $L(S)$ and $L(P)$ coincide, $L(S) = L(P)$ (or, the language $L(S)$ is contained in the language $L(P)$, $L(S) \subseteq L(P)$).*

Definition 3.17. The TFSM $\mathcal{M}_{I,O}$ over alphabets I and O is called the *maximal* TFSM if any TFSM $\mathcal{S}$ over alphabets I and O is the reduction of $\mathcal{M}_{I,O}$, i.e., $\mathcal{S} \leq \mathcal{M}_{I,O}$. △

Correspondingly, the maximal TFSM $\mathcal{M}_{I,O}$ has the language $L(\mathcal{M}_{I,O}) = [\mathtt{1}^* I \mathtt{1}^* O]^* \mathtt{1}^*$. An example of the maximal TFSM and its corresponding automaton are shown in Fig. 3.3.

Definition 3.18. The TFSM $\mathcal{S}$ with the language $L(\mathcal{S}) = \mathtt{1}^*$ is called *trivial*. △

The set of traces of trivial TFSM is empty, though, unlike the trivial classical FSM, its corresponding automaton accepts not only empty sequence, but any sequence of time symbol $\mathtt{1}$ as well.

However, not each automaton over input and output actions and action $\mathtt{1}$ has the language that is the language of some TFSM.

Proposition 3.19. *The language $L(\mathcal{S}) \subseteq L(\mathcal{M}_{I,O})$ of some TFSM $\mathcal{S}$ is $[\mathtt{1}^* \cup \mathtt{1}^* I \mathtt{1}^* O]$-prefix-closed, i.e., for any $\alpha \in L(\mathcal{M}_{I,O})$ and $b \in [\mathtt{1}^* \cup \mathtt{1}^* I \mathtt{1}^* O]$ it holds that if $\alpha b \in L(\mathcal{S})$, then $\alpha \in L(S)$ as well.*

Proof. The proposition holds due to the semantics of TFSM: all the states of TFSM are considered final, input symbols can be applied only at some states, and after producing an output symbol, the execution of transition moves TFSM to some state. □

Proposition 3.20. *Given an automaton $\mathcal{P} = \langle P, I \cup O \cup \{\mathtt{1}\}, \delta_P, p_0, Q \rangle$, with the language $L(P) \subseteq L(\mathcal{M}_{I,O})$, it corresponds to some TFSM if all the following properties hold:*

1. *The initial state is final, $p_0 \in Q$.*
2. *Each final state has an outgoing transition labeled by symbol $\mathtt{1}$, i.e., for each $p \in Q$ it holds $\mathtt{1} \in actions_P(p)$. Hence, $\mathtt{1}^* \subseteq L(\mathcal{P})$.*
3. *Each non-final state has at least one outgoing transition labeled either by symbol $\mathtt{1}$ and/or an output symbol, i.e., for each $p \in P \setminus Q$ it holds $|actions_P(p)| \geq 1$ and $actions_P(p) \subseteq O \cup \{\mathtt{1}\}$.*
4. *A transition labeled by symbol $\mathtt{1}$ can be made only between two final states or two non-final states, and there is no transition labeled by symbol $\mathtt{1}$ between a final and a non-final state, i.e., for all $\langle p_1, \mathtt{1}, p_2 \rangle \in \delta_P$ it holds either $p_1, p_2 \in Q$ or $p_1, p_2 \in P \setminus Q$.*
5. *A transition labeled with an input symbols is an outgoing transition from a final state and an incoming transition to a non-final state, i.e., for all $\langle p_1, i, p_2 \rangle \in \delta_P$ it holds $p_1 \in Q$ and $p_2 \in P \setminus Q$. For all $p \in Q$ it holds $actions_P(p) \subseteq I \cup \{\mathtt{1}\}$.*

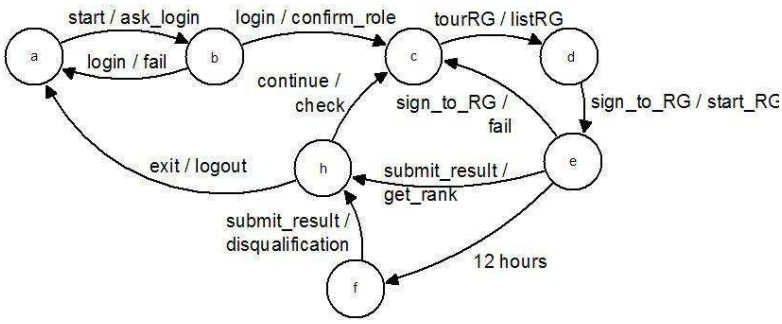

FIGURE 3.4: The sample TFSM description for Ready-Go Virtual Golf Tournament

6. *A transition labeled with an output is an outgoing transition from a non-final state and an incoming transition to a final state, i.e., for all* $\langle p_1, o, p_2 \rangle \in \delta_P$ *it holds* $p_2 \in Q$ *and* $p_1 \in P \setminus Q$*. For all* $p \in P \setminus Q$ *it holds* $actions_P(p) \subseteq O \cup \{1\}$.

If the above properties hold then we can restore a corresponding TFSM, following the Algorithm 3.3.

3.5 Web service TFSM descriptions

In this section we provide some samples of using TFSMs for modeling service behavior.

Example 3.2. *In Fig. 3.4, an example of TFSM description for participating in Virtual Golf Tournament is shown. Player signs up for participation after login in the personal account (input login), then searching for a tournament (input tourRG) that is ready to start (output listRG with available options). After choosing a tournament to participate (input* $sign_to_RG$*), the player is registered for the participation (*$start_RG$*), and allowed to participate in one tournament at a time - attempt to apply* $sign_to_RG$ *for the second time leads to a fail output. After the tournament is started, the player has 12 hours (timeout in state e) to complete the tour and submit his results, otherwise, the player is disqualified (output disqualification from state f). After the current tournament is completed, the player can continue and choose the next one to participate.*

△

Example 3.3. *Another sample TFSM, inspired by the TFSM description of Trenitalia compensation policy [73], is shown in Fig. 3.5 - the Compensator service. The service receives notifications on departure and arrival of the train, and in case of delays, the*

Algorithm 3.3 Restore TFSM from corresponding automaton

Input: The deterministic automaton $\mathcal{P} = \langle P, I \cup O \cup \{1\}, \delta_P, p_0, Q \rangle$ for which properties from Proposition 3.20 hold.

Output: The TFSM $\mathcal{S} = \langle S, I, O, s_0, \lambda_{\mathcal{S}}, \Delta_{\mathcal{S}}, \sigma_{\mathcal{S}} \rangle$ so that $Aut(\mathcal{S}) = \mathcal{P}$

$s_0 = q_0$, add s_0 to S
for each $s \in S$ **do**
Calculate timeout:
    if $\langle s, 1, s \rangle \in \delta_P$ **then** add s in S and define $\Delta_{\mathcal{S}}(s) = (s, \infty)$.
    else if $\langle s, 1, s' \rangle \in \delta_P$ **then** $t = 1$, $q = s'$
        while $s^{(t+1)} \neq s^{(k)}$ for all $0 \leq k \leq t$ where $\langle s^{(t)}, 1, s^{(t+1)} \rangle \in \delta_P$ **do**
            if for all $i \in I$ it holds $\langle s, i, f \rangle \in \delta_P$ if and only if $\langle s^{(t)}, i, f \rangle \in \delta_P$ **then**
                $q = s^{(t+1)}$ and $t = t + 1$
            end if
        end while
        Add s in S
        if $s^{(t+1)} = s^{(k)}$ for some $0 \leq k \leq t$ **then** define $\Delta_{\mathcal{S}}(s) = (s, \infty)$.
        else add q in S and define $\Delta_{\mathcal{S}}(s) = (q, t)$
        end if
    end if
Calculate transitions:
    for each $i \in actions_P(s) \cap I$ **do**
        $k = 0$, assign the set $label(p) = \{0\}$ where $\langle s, i, p \rangle \in \delta_P$
        Assign to intermediate non-final states their delays from an input:
        $q = p$
        while $\langle q, 1, f \rangle \in \delta_P$ and $|label(f)| \leq 1$ **do**
            $k = k + 1$
            if $label(f) = \emptyset$ **then** assign $label(f) = \{k\}$
            else if $|label(f)| = \{b\}$ **then** re-assign $label(f) = \{b + (k - b)t\}$
            end if
            Move $q = f$
        end while
        Use assigned delays to restore input/output transitions:
        Start from $q = p$ where $\langle s, i, p \rangle \in \delta_P$
        while $\langle q, 1, f \rangle \in \delta_P$ and $label(f) \neq \emptyset$ **do**
            for each $o \in actions_P(f) \cap O$ and $\langle f, o, s' \rangle \in \delta_P$ **do**
                if $s' \notin S$ **then** add s' to S **end if**
                if $\langle s, i, o, s' \rangle \notin \lambda_{\mathcal{S}}$ **then** add $\langle s, i, o, s' \rangle$ to $\lambda_{\mathcal{S}}$ **end if**
                Define $\sigma_{\mathcal{S}}(\langle s, i, o, s' \rangle) = \sigma_{\mathcal{S}}(\langle s, i, o, s' \rangle) \cup label(f)$
                Empty label to mark state f as explored: $label(f) = \emptyset$
            end for
            Move $q = f$
        end while
    end for
end for

reimbursement for the ticket is provided for train delays over 60 minutes (timeouts in states wait_to_depart and wait_to_arrive), if claimed within 12 month after late arrival (timeout in state late_arrival).

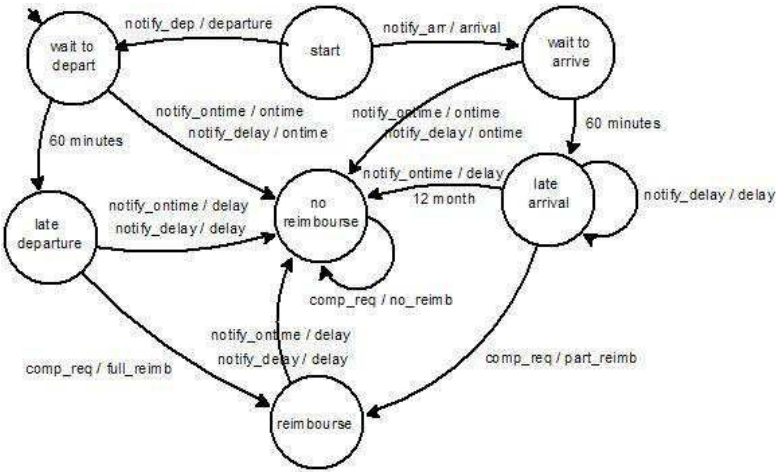

FIGURE 3.5: The TFSM description of the Compensator service

△

Example 3.4. *Let us consider a simple Loan Approval Service [74, 75], that has two components: the Assessor (Fig. 3.6) and the Approver (Fig. 3.7). The Loan Approval service is a loan bank service allowing to accept a loan for a customer or not. This service has been mainly standardized by IBM [74] but is nowadays customized and implemented by the majority of the banks in the world.*[3]

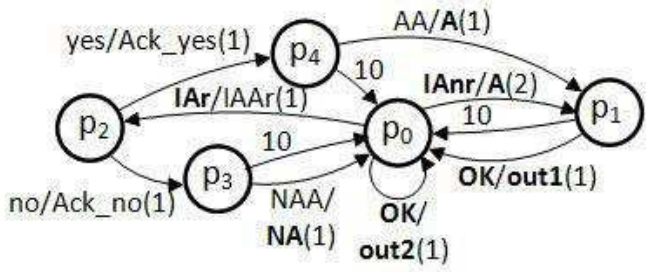

FIGURE 3.6: The Assessor component TFSM for the Loan Approval Service

First, the clients of the service send the loan request to the bank Assessor, their personal IDs and the requested amount. By using this information, the loan service may accept or refuse the loan. This decision depends on the requested amount as well as the risk linked to the client. For low amount requests (e.g., less than 10k euros) and non risky clients (input IAnr in the Fig. 3.7), the Assessor grants the loan (input A) which is provided

[3]We thank prof. Stephane Maag for this modified example of TFSM for the Loan Approval Service

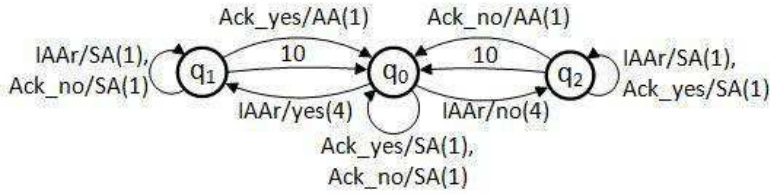

FIGURE 3.7: The Approver component TFSM for the Loan Approval Service

*(output out*1*) when the client asks for it (OK at state p*1*). In case the clients asks for a loan without assessment (OK at state p*0*), the Assessor rejects the request (output out*2*). For upper amounts or with a high risk, the Assessor forwards the loan request for additional evaluation (output IAAr). If the loan is approved by the bank (input yes), the loan is provided by the assessor after some acknowledgment phases (input AA). If the loan is not approved (input no), the request is rejected (output out*2*). In all cases, the Assessor waits for a confirmation (from the client or the additional evaluation from the bank) during 10 time units. If the confirmation is not received before timeout expires, the request is canceled.* △

3.6 Chapter conclusions

To summarize results of the chapter, we have proposed the FSM model augmented with timeout functions, relying on single unique time variable, synchronized with the global timer of the environment/user, for description of web service behavior integrating time-related quality parameters in output delays function. For the proposed model, which is a time-nondeterministic extension of the one known from the related works, we have established and investigated conformance relations with respect to different value domain for the time variable, and showed that all conformance relations in consideration are preserved when restricting the values of time variable form real to integer domain. The latter allows to establish sound correspondence between TFSMs and Finite Automata and to solve many TFSM problems using well-developed automata theory methods and tools.

Some discussion on the seemingly-limited expressive power of TFSM model for service quality evaluation, taking into account only time parameter, is provided in Section 4.5, where we point out that all the obtained results for TFSM to FA correspondence as well

as composition could be directly applied to the cases when time variable of the TFSM is interpreted as any other cumulative quality parameter.

Chapter 4

Parallel Composition of TFSMs as a Service Composition

4.1 Introduction

Complex services are often derived as a composition of already existing ones. In order to formally impose the restrictions on functional and non-functional service parameters which are described by TFSMs, in this chapter we define the parallel composition operator and discuss safety issues that may appear in the communication. Since the behavior of component services involved in the composition might be significantly nondeterministic and partial due to lack of observability and control of component service side communications with other services/users/applications outside the scope of the one in consideration, the distinctive discussion on the composition of partial nondeterministic machines need to be done.

4.2 Parallel composition as a service composition

The general topology for the parallel composition is shown in Fig. 4.1. In this thesis we consider only binary composition, though, in many cases it can be hierarchically extended to the arbitrary number of components.

We assume, that components communicate in so-called *slow environment*, i.e., the next external request can be applied only after the response to the previous one has been

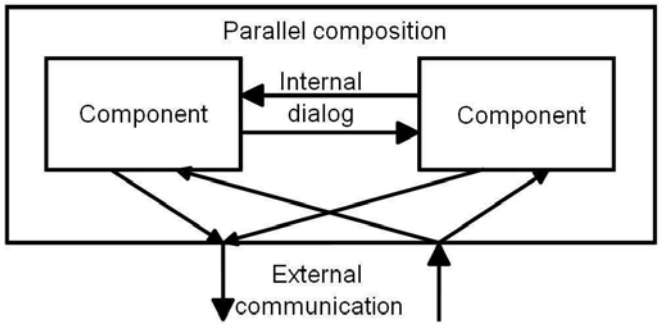

FIGURE 4.1: General topology of the parallel composition

received. In other words, we assume that communication starts with receiving the request, ends with producing the response, and in the process of producing the response only relevant internal dialog is considered. In case of multiple users, only one process is considered at each moment of time, enclosing possible influences of other communications in nondeterministic behavior of components.

The communication is organized in a following way: after the composition receives the external request from the environment (e.g., service user or some other services), the request is processed by the responsible component, and either an external response is produced, or an internal request to another component is sent. Another component processes the request and either produces an external response, or sends an internal request to the first component. The internal dialog proceeds until the external response is produced. All internal communications are not observable from the environment (or by users).

For service composition, there are two main concepts widely discussed in the literature: orchestration and choreography. Orchestration is considered to be centralized composition with the dedicated service (called orchestrator) to manage all the interactions in the system and be a mediator between the user of the service and components involved in producing the final result. Choreography, on the contrary, is considered as distributed decentralized composition, with no dedicated service to organize collaborative work, with each involved agent knowing when and with whom to engage in interaction, and with possibility of moving the user-interaction point from one component to another according to composition goals. Different from orchestration, choreography does not necessarily describe internal actions and internal (control) message exchanges between components, primarily sequences of messages that are observable from global point of

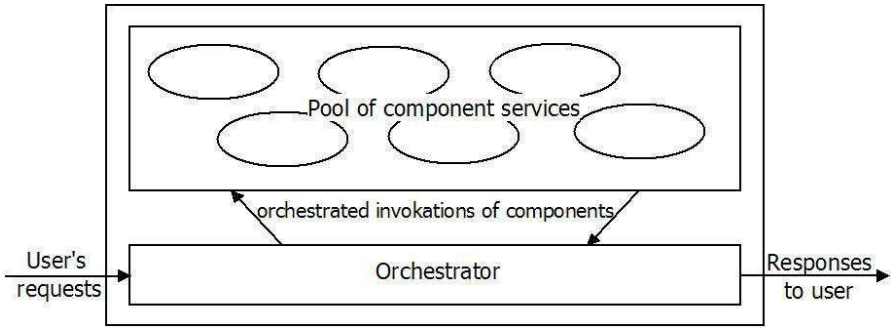

FIGURE 4.2: Service orchestration in form of parallel composition

view. Often [17, 18] choreography is considered as a communication protocol to which the collaborative behavior of individual peers (component services) has to conform.

In our work, we omit the discussion of implementation details, whether the composition is centralized or not. The crucial point of parallel composition concept lays in request-response mode of communication of the composition with the environment (or, user) and between components. Therefore, the adaptation of such composition to common service composition concepts, might be done, for example, as follows.

In orchestration, the communication of user with the service goes through the orchestrator, and all other components can be considered as an embedded pool of components (Fig.4.2) to which the orchestrator makes requests according to the external one from a user. In choreography, the communication with user might be switched from one to another component service, i.e., all components might receive and/or produce external requests and responses, and the internal non-observable communications are dedicated to the invocation and confirmation requests between components.

More complex and detailed consideration of orchestration and choreography, including the representation of distributed behavior with simultaneous work of several components, is a question for future work.

4.3 TFSMs parallel composition

The behavior of a service can be described by sequences of inputs (requests), with time stamps of applying, followed by outputs (responses), with time stamps of receiving, therefore, represented by a TFSM.

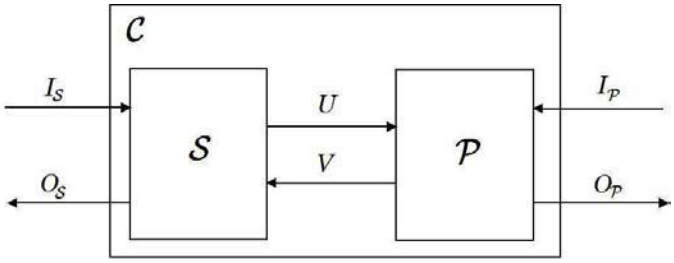

FIGURE 4.3: Parallel composition of the TFSMs $\mathcal{S}$ and $\mathcal{P}$

For each user, the service (TFSM) parallel composition can be described in the following way. Consider the composition in Fig. 4.3, where $I = I_{\mathcal{S}} \cup I_{\mathcal{P}}$ and $O = O_{\mathcal{S}} \cup O_{\mathcal{P}}$ are the external input and output alphabets of composition, U and V are internal alphabets, in which the dialog between components is performed. Each component and composition have their own independent timers, which increase their values with the same speed synchronized with global time, but are independently reset according to transitions and timeouts happening in each component. Composition is considered to be in its initial state when both components are in their initial states and all timers are reset to 0.

The work of composition is initiated by the request via $I_{\mathcal{S}}$ or $I_{\mathcal{P}}$. Without loss of generality, suppose, that the request is applied via $I_{\mathcal{S}}$. Then, the service $\mathcal{S}$ processes the request, and, after some delay, either replies with the external response via $O_{\mathcal{S}}$, or sends an internal request to $\mathcal{P}$ via U. In both cases, while $\mathcal{S}$ processes applied input, the component $\mathcal{P}$ might change its state according to the timeouts. After receiving the request from $\mathcal{S}$ via U, the component $\mathcal{P}$ processes it, and either produces an external response via $O_{\mathcal{P}}$ or replies to $\mathcal{S}$ via V. The internal dialog continues until one of the components produces the external output (response). After that, the composition is ready to receive and process next external input (request).

The dialog between components should be finite and result into some external output, otherwise, unsafe behavior of the composition might be observed: the absence of livelocks (infinite internal dialogs) and deadlocks (impossibility of continuing any dialog) is one of the crucial safety requirements for communicating components. Safety-awareness of composition is considered further in Section 4.4.3. As we notice, the opportunity to deal with a number of users is modeled by functional and time non-determinism in the components. The assumption of non-zero minimal output delay might be useful for safety-aware composition, because in this case, each internal dialog has non-zero length

in time and, hence, the livelocks can be detected by exceeding of expected maximal delay in the composition.

4.3.1 Formal definition of parallel composition

According to the above description, the behavior of the parallel composition of two TFSMs can be described based on the automata which correspond to component TFSMs (Section 3.4.2). However, the language of the parallel composition of two automata is not necessarily a TFSM language. For this reason, the obtained language should be intersected with the language of the maximal TFSM $\mathcal{M}_{I,O}$, where I and O are external input and output alphabets of composition, to ensure that each input is followed by some output.

Proposition 4.1. *If $\mathcal{S}$ and $\mathcal{P}$ are TFSMs then the language of the automaton $Aut(\mathcal{S})\Diamond Aut(\mathcal{P}) \cap Aut(\mathcal{M}_{I,O}) = \left(Aut(\mathcal{S})_{\Uparrow I_P \cup O_P} \cap Aut(\mathcal{P})_{\Uparrow I_S \cup O_S}\right)_{\Downarrow I \cup O \cup \{1\}} \cap Aut(\mathcal{M}_{I,O})$ accepts a TFSM language.[76]*

The automaton $Aut(\mathcal{S})\Diamond Aut(\mathcal{P}) \cap Aut(\mathcal{M}_{I,O})$ then can be transformed into a TFSM.

Definition 4.2. Given the TFSMs $\mathcal{S}$ and $\mathcal{P}$, the TFSM $\mathcal{C} = \mathcal{S}\Diamond\mathcal{P}$ with the corresponding automaton $Aut(\mathcal{C}) = [Aut(\mathcal{S})\Diamond Aut(\mathcal{P})] \cap Aut(\mathcal{M}_{I,O}) = \left(Aut(\mathcal{S})_{\Uparrow I_P \cup O_P} \cap Aut(\mathcal{P})_{\Uparrow I_S \cup O_S}\right)_{\Downarrow I \cup O \cup \{1\}} \cap Aut(\mathcal{M}_{I,O})$ is called the *parallel composition* of $\mathcal{S}$ and $\mathcal{P}$. △

4.3.2 Composition traces: external, internal and global

Given the component TFSMs $\mathcal{S} = \langle S, I_S \cup V, O_S \cup U, s_0, \lambda_\mathcal{S}, \Delta_\mathcal{S}, \sigma_\mathcal{S}\rangle$ and $\mathcal{P} = \langle P, I_P \cup U, O_P \cup V, p_0, \lambda_\mathcal{P}, \Delta_\mathcal{P}, \sigma_\mathcal{P}\rangle$ with corresponding finite automata $Aut(\mathcal{S})$ and $Aut(\mathcal{P})$, respectively, denote the external alphabets of composition $I = I_S \cup I_P$, $O = O_S \cup O_P$, for short.

Definition 4.3. An automaton $Globe(\mathcal{S}\Diamond\mathcal{P}) = (Aut(\mathcal{S})_{\Uparrow I_P \cup O_P} \cap Aut(\mathcal{P})_{\Uparrow I_S \cup O_S})$ is called a *global automaton of the composition* $\mathcal{C} = \mathcal{S}\Diamond\mathcal{P}$ (or simply a *global automaton Globe* if there is no ambiguity). △

Definition 4.4. Given a composition $\mathcal{C} = \mathcal{S}\Diamond\mathcal{P}$, the following types of traces are associated with it:

- *Global traces* of the composition are the strings accepted by the global automaton and representing the communication process between components started by applying external inputs and tailed by external outputs, i.e., the traces $\nu \in L(Globe)$ so that $\nu \in (\mathtt{1}^*I(U \cup V \cup \{\mathtt{1}\})^*O)^*$.

- *External traces* of the composition are the strings accepted by the automaton corresponding to the composition TFSM, i.e., γ is an external trace if $\gamma \in L(\mathcal{C}) = L(Globe)_{\Downarrow I \cup O \cup \{\mathtt{1}\}} \cap L(\mathcal{M}_{I,O})$.

- *Internal traces* of the composition are the strings ω of the set $\nu_{\Downarrow U \cup V \cup \{\mathtt{1}\}} \in (U \cup V \cup \{\mathtt{1}\})^*$ where $\nu \in L(Globe) \cap (\mathtt{1}^*I(U \cup V \cup \{\mathtt{1}\})^*O)^*$ is a global trace of the composition, i.e., internal traces represent internal dialogs between components. △

Remark 4.5. Since for web service composition, as mentioned, the particular case of the composition with embedded component is often useful, we note, that if one of the component is embedded (i.e., its external alphabets are empty) and all the external communications are made via another component (context), then the context component contains all the possible global traces of the composition. △

External and global composition traces have the following relationship.

Proposition 4.6. *A trace $\gamma \in (\mathtt{1}^*I\mathtt{1}^*O)^*$ is an external trace of the composition if and only if there exists a global trace $\nu \in (\mathtt{1}^*I(U \cup V \cup \{\mathtt{1}\})^*\mathtt{1}^*O)^*$ such that $\gamma = \nu_{\Downarrow I \cup O \cup \{\mathtt{1}\}}$.*

Proof. By the definition of parallel composition and global automaton, it holds $Aut(\mathcal{C}) = Globe(\mathcal{C})_{\Downarrow I \cup O \cup \{\mathtt{1}\}} \cap Aut(\mathcal{M}_{I,O})$.

⇒ Consider a trace $\nu \in (\mathtt{1}^*I(\mathtt{1}^*U\mathtt{1}^*V)^*\mathtt{1}^*O)^*$ being a global trace $\nu \in L(Globe)$, then $\nu_{\Downarrow I \cup O \cup \{\mathtt{1}\}} \in L(Globe)_{\Downarrow I \cup O \cup \{\mathtt{1}\}}$ and $\nu_{\Downarrow I \cup O \cup \{\mathtt{1}\}} \in (\mathtt{1}^*I\mathtt{1}^*O)^*$, which means that $\nu_{\Downarrow I \cup O \cup \{\mathtt{1}\}} \in L(Globe)_{\Downarrow I \cup O \cup \{\mathtt{1}\}} \cap L(\mathcal{M}_{I,O})$. As the latter is the language of the automaton $Aut(\mathcal{C})$, the $\nu_{\Downarrow I \cup O \cup \{\mathtt{1}\}}$ is an external trace of the composition.

⇐ Consider an external trace $\gamma \in (\mathtt{1}^*I\mathtt{1}^*O)^*$ that is a string in the language of the automaton $Aut(\mathcal{C}) = Globe_{\Downarrow I \cup O \cup \{\mathtt{1}\}} \cap Aut(\mathcal{M}_{I,O})$. Due to the properties of operations over finite automata and regular languages [71], since $[L(\mathcal{M}_{I,O})_{\Uparrow U \cup V}]_{\Downarrow I \cup O \cup \{\mathtt{1}\}} = L(\mathcal{M}_{I,O})$ it holds that $Aut(\mathcal{C}) = [Globe \cap Aut(\mathcal{M}_{I,O})_{\Uparrow U \cup V}]_{\Downarrow I \cup O \cup \{\mathtt{1}\}}$, i.e.,

$\gamma \in [L(Globe) \cap L(\mathcal{M}_{I,O})_{\Uparrow U \cup V}]_{\Downarrow I \cup O \cup \{1\}}$ and there exists at least one (global) trace ω, $\omega \in L(Globe)$ and $\omega \in L(\mathcal{M}_{I,O})_{\Uparrow U \cup V}$, and $\gamma = \omega_{\Downarrow I \cup O \cup \{1\}}$.

□

Consider an external trace $\gamma = 1^{t_1} i_1 1^{k_1} o_1 \dots 1^{t_n} i_n 1^{k_n} o_n$, denote $\gamma_{\downarrow I} = 1^{t_1} i_1 \dots 1^{t_n} i_n$ the external input sequence corresponding to the timed input sequence $\alpha = \langle i_1, t_1 \rangle \dots \langle i_n, t_n \rangle$ of the composition, and $\gamma_{\downarrow O} = 1^{k_1} o_1 \dots 1^{k_n} o_n$ the external output sequence corresponding to the timed output sequence $\beta = \langle o_1, k_1 \rangle \dots \langle o_n, k_n \rangle$ of the composition. For the simplicity of notations, denote $\gamma = \alpha\beta$.

Definition 4.7. An external timed input sequence α *induces* a global trace ν, if there exists an external timed output sequence β such that $\alpha\beta = \nu_{\Downarrow I \cup O \cup \{1\}}$, and in this case the global trace ν is said to *support* an external trace $\alpha\beta$.

A global trace ν is said to *violate* an external trace $\alpha\beta$ if ν is induced by α and $\alpha\beta \neq \nu_{\Downarrow I \cup O \cup \{1\}}$, i.e., there exist $\beta' \neq \beta$ such that $\alpha\beta' = \nu_{\Downarrow I \cup O \cup \{1\}}$. △

Definition 4.8. Given an external timed input sequence α and an internal trace γ, α *induces* an internal trace γ if α induces a global trace ν such that $\gamma = \nu_{\Downarrow U \cup V \cup \{1\}}$; in this case, the internal trace γ is said to *support* the external trace $\alpha\beta = \nu_{\Downarrow I \cup O \cup \{1\}}$. An internal trace γ is said to *violate* an external trace $\alpha\beta$ if $\gamma = \nu_{\Downarrow U \cup V \cup \{1\}}$ is induced by α and the global trace ν violated $\alpha\beta$. △

The set of all global traces of the composition that support given external trace $\alpha\beta$ is denoted $sup^{global}(\alpha\beta) = L(Global) \cap (\alpha\beta)_{\Uparrow U \cup V}$.

The set of all internal traces of the composition that support given external trace $\alpha\beta$ is denoted $sup^{internal}(\alpha\beta) = sup^{global}(\alpha\beta)_{\Downarrow U \cup V \cup \{1\}}$.

Proposition 4.9. *Given an external trace $\alpha\beta$ and an internal trace γ, γ violates γ if α induces a global trace ν such that $\nu_{\Downarrow I \cup O \cup \{1\}} = \alpha\beta'$ with $\nu_{\Downarrow U \cup V \cup \{1\}} \in \gamma 1^*$ while $\beta' \neq \beta$.*

Proof. This proposition holds due to the definition of violating internal traces and properties of the restriction and expansion operators. A suffix 1^* after an internal trace indicates when the next internal symbol may appear after the next external input, and, by definition of the TFSM and corresponding automata, for all traces γ it holds that $\gamma 1^*$ are also traces of corresponding automaton. □

4.3.3 TFSMs closure properties under parallel composition

Proposition 4.10. *The set of deterministic TFSMs is closed under parallel composition.[76]*

However, if TFSMs $\mathcal{S}$ and/or $\mathcal{P}$ are t-deterministic but f-nondeterministic, then, in general case, the composition $\mathcal{S}\Diamond\mathcal{P}$ can be both t- and f-nondeterministic.

Proposition 4.11. *The set of time-deterministic TFSMs is not closed under parallel composition.[76]*

Proof. We prove this statement with a counter-example in Fig. 4.4. Both components $\mathcal{S}$ and $\mathcal{P}$ are t-deterministic, while the component $\mathcal{S}$ is f-nondeterministic. Therefore, in the global automaton of the composition of $\mathcal{S}\Diamond\mathcal{P}$ there exist two different traces induced by the external input symbol i and leading to producing the external output y: $i1u11v1y$ and $i11w111v1y$, both taking the global automaton (Fig. 4.4b) from initial state to initial state. The latter cause the composition TFSM (Fig. 4.4c) to have two different output delays values for the same loop transition in the state $\langle s, a\rangle$ under input-output pair i/o.

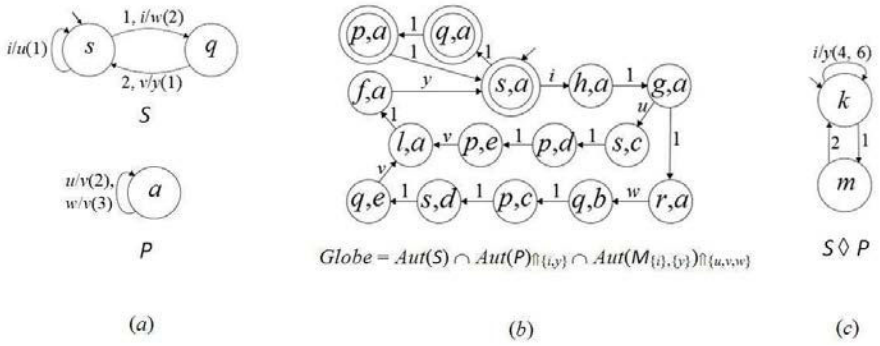

FIGURE 4.4: An example of t-nondeterministic composition (c) of t-deterministic components $\mathcal{S}$ and $\mathcal{P}$ (a) with corresponding global automaton (b)

□

Proposition 4.12. *The set of TFSMs is closed under parallel composition.[76]*

Proof. The proposition holds due to the Proposition 4.1 and closure properties of finite automata. Any sub-graph of a finite automaton is a finite automaton, therefore, any

group of transitions by special symbol 1 between transition by input symbol and transition by output symbol, is a finite automaton and can be represented with a set of linear functions. Therefore, any output delay even in case of nondeterministic components can be expressed in form $\{\{kt + b\}|k, b \in \mathbb{N}\}$. □

Definition 4.2 returns only the enabled behavior of parallel composition, with no consideration of safety and robustness properties. If any livelock without a possibility of producing an external output appears, or a deadlock (i.e., if during the communication one of the components attempts to apply an input to another component that is not defined in it - and, hence, with catastrophic interpretation may cause damage to the system [77]), such sequences will be just cut off by restriction, intersection with maximal compositional automaton and resulting TFSM retrieval operations. If some livelock in composition of nondeterministic components may lead to the producing of external output, it will be relflected by the periodic output delay in corresponding transition in the resulting TFSM.

For complete components, either deterministic or not, the given definition is representative enough, since if the resulting TFSM has some input sequences undefined, it directly indicates the livelock (internal infinite cycle without producing an external output) and such sequences could be just forbidden as unsafe.

Example 4.1. *As an example of parallel composition, let us derive the composition of the Assessor and the Approver components of the Loan Approval Service from the Fig. 3.7 and Fig. 3.6. The resulting composed service is shown in Fig. 4.5.*

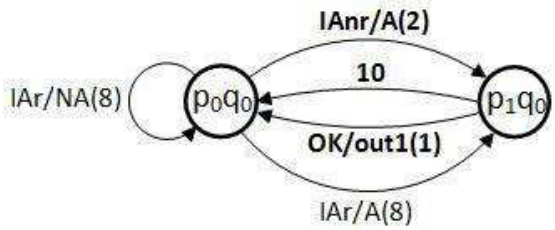

FIGURE 4.5: The composed TFSM for Loan Approval Service

△

However, for partial components, if resulting composition TFSM has undefined input sequences, the reasons may vary: undefined because of components partiality - and no

addition control needed there, - or because livelock or deadlock are induced, and the latter may lead to unpredictable behavior of the component including system crush [77].

Since web services models are essentially nondeterministic and partial, it would be too arrogant and insufficient to consider the parallel composition derivation that is representative for complete components only.

Therefore, in the next section the safety issues of the composition are considered and the algorithms of verifying the safety of composition are proposed.

The general language-based definition of parallel composition does not capture some features and properties valuable for the compositions of partial and nondeterministic components. In particular, language-based parallel composition of TFSMs fails to resolve compositional safety and interoperability issues without additional control. For complete deterministic components, occurrence of livelocks results in composition TFSM being partial, while for partial and/or nondeterministic components partiality of composition TFSM may or may not be caused by any live- or deadlock in components communication.

4.4 Safe TFSM composition

Definition 4.13. Given the TFSMs $\mathcal{S}$ and $\mathcal{P}$, the parallel composition $\mathcal{C} = \mathcal{S} \lozenge \mathcal{P}$ is *safe* if it never falls neither into livelock nor deadlock.

Given the TFSMs $\mathcal{S}$ and $\mathcal{P}$, the parallel composition $\mathcal{C} = \mathcal{S} \lozenge \mathcal{P}$ is said to fall into a *livelock* if the components $\mathcal{S}$ and $\mathcal{P}$ induce an infinite internal dialog without producing any external output, i.e., if the global automaton of the composition has cycles labeled with internal and time symbols only.

Given the TFSMs $\mathcal{S}$ and $\mathcal{P}$, the parallel composition $\mathcal{C} = \mathcal{S} \lozenge \mathcal{P}$ is said to fall into a *deadlock* if neither the producing of the external output nor continuation of the internal dialog between components is possible, i.e., if the global automaton of the composition has reachable non-final states with no outgoing transitions. △

Throughout this thesis we adopt catastrophic interpretation for either livelocks or deadlocks, meaning, that if there exists a possibility of composition to produce unsafe behavior, the unsafe behavior will be eventually produced. In more details, for livelock it

means that if the global automaton has a cycle label by internal and time symbols only, but this cycle has an exit by an external output (nondeterministic choice of whether to leave or continue the cycle is in place), then we still consider such situation as livelock-unsafe.

Deadlocks in composition may appear in case of partial components, if one component tries to apply an internal symbol to the another component that is not allowed in the another component - and, hence, the continuation of the communication between components becomes unpredictable and/or impossible.

The interpretation of partiality in TFSMs can vary, two major types being 1) don't care sequences (allowing any possible behavior for undefined input sequences), and 2)forbidden behavior (the threat of unpredictable catastrophic consequences of applying undefined input sequences[77], or physical impossibility of applying). The first type (don't care) is solved by explicit definition of don't care behavior, therefore, we further assume that all partiality in component TFSMs are of second type (forbidden sequences). We remind, though, that in any case we consider TFSMs with harmonized traces only.

4.4.1 Livelock-safe composition

Since we use non-determinism for modeling the behavior due to queries via other channels (which are not considered in our model) we consider the crucial interpretation of livelocks. In other words, if an input sequence induces a cycle of internal actions (and, possibly, a timed symbol) then this sequence has to be forbidden, i.e., has to be excluded from the corresponding composed TFSM.

Similar to the classical FSMs, one way of restricting livelock possibility is to restrict the length of internal dialogs between components [71]. If the internal communication exceed the length bound, presence of the livelock might be suspected and corresponding input sequence is forbidden in the composition. This approach can be adapted to TFSMs.

Length-bounded composition

Length of the internal dialog is a number of internal actions the components exchange before producing an external output. In the general definition of parallel composition,

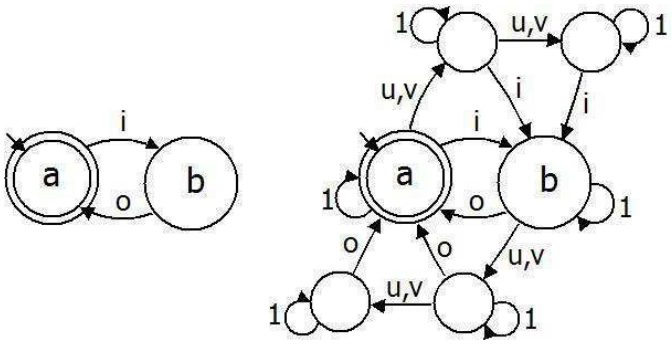

FIGURE 4.6: The automata $Aut(I,O)$ (left) and $(Aut(I,O)_{\Uparrow U\cup V,l})_{\Uparrow\{1\}}$ (right) with alphabets $I=\{i\}$, $O=\{o\}$, $U=\{u\}$, $V=\{v\}$ and length bound $l=2$

the length of internal dialogs is not bounded.

Let $l \in \mathbb{N}$ be the length bound, denote $L_{\Uparrow U\cup V,l}$ the *l-expansion* of L, i.e., the the language $(L_{\Uparrow U\cup V,l})_{\Downarrow U\cup V}$ has words no longer than l. Denote $Aut(I,O)$ the automaton with language $(IO)^*$. Then, the $Aut(I,O)_{\Uparrow U\cup V,l}$ is the automaton with all possible communications between components with the internal dialogs not longer than l, while $(Aut(I,O)_{\Uparrow U\cup V,l})_{\Uparrow\{1\}}$ represents all possible length-bounded communications with arbitrary delays between symbols.

Definition 4.14. Given the TFSMs $\mathcal{S}$ and $\mathcal{P}$, the *l-bounded* parallel composition is the TFSM $\mathcal{C} = \mathcal{S}\Diamond_l\mathcal{P}$ with corresponding automaton $Aut(\mathcal{C}) = [Aut(\mathcal{S})\Diamond_l Aut(\mathcal{P})] \cap Aut(\mathcal{M}_{I,O}) = [Aut(\mathcal{S})_{\Uparrow I_P\cup O_P} \cap Aut(\mathcal{P}_{\Uparrow I_S\cup O_S}) \cap (Aut(I,O)_{\Uparrow U\cup V,l})_{\Uparrow\{1\}}]_{\Downarrow I\cup O\cup\{1\}} \cap Aut(\mathcal{M}_{I,O})$. △

The automaton $(Aut(I,O)_{\Uparrow U\cup V,l})_{\Uparrow\{1\}}$ (shown in Fig. 4.6 for a simple case) represents all possible communication between components such that there are no more than l internal symbols $U\cup V$ in each dialog, with no restriction on the time length.

However, in practice, measuring time between requests and responses is more applicable than observing and counting internal actions, and transition from length-bounded composition to time-bounded composition might be useful.

Time-bounded composition

In general case, we do not restrict output delays in components, though, often in practice the measure of one time unit is chosen in such way that the minimal output delay is non-zero.

Using properties of restriction and expansion operations [71], let us move $Aut(\mathcal{M}_{I,O})$ inside the brackets, i.e., we re-write the composition in form

$$Aut(C) = [Aut(\mathcal{S})_{\Uparrow I_P \cup O_P} \cap Aut(\mathcal{P})_{\Uparrow I_S \cup O_S} \cap (Aut(I,O)_{\Uparrow U \cup V,l})_{\Uparrow \{1\}} \cap Aut(\mathcal{M}_{I,O})_{\Uparrow U \cup V}]_{\Downarrow I \cup O \cup \{1\}},$$

and consider the intersection $(Aut(I,O)_{\Uparrow U \cup V,l})_{\Uparrow \{1\}} \cap Aut(\mathcal{M}_{I,O})_{\Uparrow U \cup V}$. From

$$L(I,O)_{\Uparrow U \cup V,l})_{\Uparrow \{1\}} = ((IO)^*_{\Uparrow U \cup V,l})_{\Uparrow \{1\}},$$

and

$$L(M_{I,O})_{\Uparrow U \cup V} = [(1^* I 1^* O)^* 1^*]_{\Uparrow U \cup V},$$

we conclude

$$L(I,O)_{\Uparrow U \cup V,l})_{\Uparrow \{1\}} \cap L(M_{I,O})_{\Uparrow U \cup V} = (1^* I [(U \cup V)^{\leq l}]_{\Uparrow \{1\}} 1^* O)^* 1^*$$

.

Let us denote $m, n \in \mathbb{N}$ the minimal and maximal output delays in component $\mathcal{S}$, respectively.

In any sequence α from $L(\mathcal{S})_{\Uparrow I_P \cup O_P} \cap L(I,O)_{\Uparrow U \cup V,l})_{\Uparrow \{1\}} \cap L(M_{I,O})_{\Uparrow U \cup V}$ there are at most l symbols from $U \cup V$ between each input and output symbol, and since in $L(\mathcal{S})_{\Uparrow I_P \cup O_P}$ there are at most n symbols 1 between any two neighbor symbols, then in any α between input and output symbols there are at most $n(l+1)$ symbols 1.

In other words, we transform length-bounded composition to time-bounded composition:

$$L(\mathcal{C}) = [L(\mathcal{S}) \Diamond L(\mathcal{P})] \cap [1^* I 1^{[m;n(l+1)]} O]^* 1^*.$$

In case at least one of the components is defined with minimal non-zero output delay, there is no internal cycle with zero time-length possible, i.e., all possible livelocks have non-zero duration.

Therefore, based on length-bound for composition and knowing the maximal output delay for components, we can calculate the maximal output delay for the bounded composition.

The explicit length-bound, then, might be omitted and completely substituted with timed-bound.

Time-bound is quite a practical approach; for classical FSMs, for example, the detection of livelocks is based on setting the maximal waiting time. The difference of TFSMs from FSMs, for detecting livelocks, lays in possibility to calculate this maximal time (time-bound) based on the delays defined in components, and, hopefully, set more adequate maximal waiting time.

Definition 4.15. Given TFSMs $\mathcal{S}$ and $\mathcal{P}$, the *t-bounded* parallel composition is the TFSM $\mathcal{C} = \mathcal{S} \Diamond_t \mathcal{P}$ with corresponding automaton $Aut(\mathcal{C}) = [Aut(\mathcal{S}) \Diamond Aut(\mathcal{P})] \cap Aut(\mathcal{M}_{I,O,t})$ where $L(\mathcal{M}_{I,O,t}) \subseteq (\mathtt{1}^* I (\mathtt{1})(\mathtt{1}^{\leq t}) O)^* \mathtt{1}^*$. △

Remark. The timed-bounded composition has two bounds: minimal and maximal output delay, and, in general, does not necessarily imply the length bound on communication. The minimal bound is used to cut out from composition all livelocks with zero duration. However, by practical reasons, it is worth to use either the assumption of non-zero minimal output delay in components, or non-zeno runs assumption from timed automata [78] implying that in a finite time interval there could not happen infinite number of actions. For parallel composition, the latter results in the requirement, that though some transitions might have zero output delays, there are no cycles with zero duration possible. In other words, all cycles in global automaton must contain at least one symbol $\mathtt{1}$. The maximal bound is useful to cut out livelocks including the ones with periodic output delays and possibility to produce an external output. Therefore, in case of livelock-unsafe composition, t-bounded composition might be used efficiently to refine the composition specification, and, using results of Chapter 5 and Chapter 6, optimize components to meet safety requirements. △

4.4.2 Deadlock-safe composition

In case of partial component TFSMs, we assume, that any undefined external input sequences is never applied to the composition. Then, deadlocks occur in the composition if one component attempts to apply an undefined symbol to another component.

Therefore, to verify and ensure deadlock-safety of the composition, it is essential to determine the allowed internal input sequences for each component.

Given a partial component TFSM $\mathcal{S}$, the set of all internal input sequences accepted by $\mathcal{S}$ defines the set of allowed internal output sequences for another component $\mathcal{P}$. If $\mathcal{P}$ never produces internal sequence out of allowed set, then it never causes deadlock in the component $\mathcal{S}$. If another component produces output sequences only from the allowed set, then it never causes any deadlocks to ensure deadlock-free composition, and vice-versa. So, given the component TFSM, it is possible to derive the maximal TFSM for another component that communicates deadlock-free with the given component.

Algorithm 4.1 Deriving the maximal TFSM for deadlock-safe component

Input: The component TFSM $\mathcal{S} = \langle S,\ I_S \cup V,\ O_S \cup U,\ s_0,\ \lambda_\mathcal{S},\ \Delta_\mathcal{S}, \sigma_\mathcal{S}\rangle$
Output: The maximal TFSM $\mathcal{M}^\mathcal{S}_{U,V}$ such that $\mathcal{S} \Diamond \mathcal{M}^\mathcal{S}_{U,V}$ is deadlock-free

Derive a finite automaton that represents the set of accepted timed input sequences $in_S(s_0)$:

1: Derive $\mathcal{S}' = \langle S,\ I_S \cup V, \{\epsilon\},\ s_0,\ \lambda'_\mathcal{S},\ \Delta_\mathcal{S}, \sigma'_\mathcal{S}\rangle$ so that:
2: **for** each transition $\langle s_1, i, o, s_2\rangle \in \lambda_\mathcal{S}$ **do**
3: there is a transition $\langle s_1, i, \epsilon, s_2\rangle \in \lambda'_\mathcal{S}$
4: with output delay function value $\sigma'_\mathcal{S}(\langle s_1, i, \epsilon, s_2\rangle) = \{0\}$
5: **end for**
6: Derive $Aut(\mathcal{S}')$ using Algorithm 3.2.

Leave external input sequences out of consideration:

7: Restrict $Aut(\mathcal{S}')_{\Downarrow V \cup \{1\}}$ to represent accepted internal input sequences.

Inverse internal input symbols to become internal output symbols:

8: Expand $Aut(\mathcal{S}')_{\Uparrow U}$.
9: Intersect $Aut(\mathcal{M}) = Aut(\mathcal{S}')_{\Uparrow U} \cap Aut(\mathcal{M}_{U,V})$ where $L(\mathcal{M}_{U,V}) = (1^*U1^*V)^*1^*$.

Result:

10: Restore $Aut(\mathcal{M})$ into TFSM $\mathcal{M}^{\mathcal{S}_{U,V}}$ using the Algorithm 3.3.

The result of Algorithm 4.1 is the maximal TFSM that contains all the allowed internal sequences that do not induce a deadlock in the given component.

To ensure the deadlock-free composition, such maximal TFSMs should be derived for both components. If the internal part of first component's behavior is contained in the maximal TFSM derived by the Algorithm 4.1 from the second component, then the first component never apply any undefined sequences to the second component.

Given two components, that might have deadlocks in communication, based on the results of the Algorithm 4.1 it is possible to derive their largest reductions which communicate without deadlocks, as proposed in Algorithm 4.2. Though, there is no guarantee that the derived deadlock-safe composition will be non-trivial. If Algorithm 4.1 for one of the components return an empty result, it means that the given components cannot communicate in a deadlock-safe manner, and such composition should be re-designed.

Algorithm 4.2 Deriving the deadlock-safe reductions of given components

Input: The component TFSMs $\mathcal{S} = \langle S, I_S \cup V, O_S \cup U, s_0, \lambda_{\mathcal{S}}, \Delta_{\mathcal{S}}, \sigma_{\mathcal{S}} \rangle$ and $\mathcal{P} = \langle P, I_P \cup U, O_P \cup V, p_0, \lambda_{\mathcal{P}}, \Delta_{\mathcal{P}}, \sigma_{\mathcal{P}} \rangle$

Output: The component TFSMs $\mathcal{S}^{ds} \leq \mathcal{S}$ and $\mathcal{P}^{ds} \leq \mathcal{P}$ so that $\mathcal{S}^{ds} \Diamond \mathcal{P}^{ds}$ is deadlock-free

1: Derive $\mathcal{M}^{\mathcal{S}}_{U,V}$ and $\mathcal{M}^{\mathcal{P}}_{V,U}$ using Algorithm 4.1.
2: Derive $Aut(\mathcal{M}^{\mathcal{S}}_{U,V})$ and $Aut(\mathcal{M}^{\mathcal{P}}_{V,U})$.
3: Derive $Aut(\mathcal{M}^{\mathcal{S}}_{U,V})_{\Uparrow I_P \cup O_P} \cap Aut(\mathcal{M}_{I_P \cup U, O_P \cup V}) = Aut(M^{ds}_P)$.
4: Derive $Aut(\mathcal{M}^{\mathcal{P}}_{V,U})_{\Uparrow I_S \cup O_S} \cap Aut(\mathcal{M}_{I_S \cup V, O_S \cup U}) = Aut(M^{ds}_S)$.
5: Intersect $Aut(\mathcal{P}^{ds}) = Aut(\mathcal{P}) \cap Aut(M^{ds}_P)$.
6: Intersect $Aut(\mathcal{S}^{ds}) = Aut(\mathcal{S}) \cap Aut(M^{ds}_S)$.
7: Restore TFSMs $\mathcal{P}^{ds}$ - the largest reduction of $\mathcal{P}$ which does not induce any undefined behavior in $\mathcal{S}$.
8: Restore TFSMs $\mathcal{S}^{ds}$ - the largest reduction of $\mathcal{S}$ which does not induce any undefined behavior in $\mathcal{P}$.
9: **if** $\mathcal{S}^{ds}$ and $\mathcal{P}^{ds}$ are non-trivial **then**
10: $\mathcal{S}^{ds} \Diamond \mathcal{P}^{ds}$ is deadlock-free.
11: **else**Deadlock-free communication between given components is impossible.
12: **end if**

4.4.3 Deriving safe parallel composition

Verification whether the composition of given components is safe or not may be done during the composition derivation, aborting the process with the verdict "unsafe"as soon as a livelock loop, labeled with internal and time symbols only, or deadlock state, non-final state with no outgoing transitions, are reached in global automaton.

Another way, though, is to preserve in the resulting composition machine all the safety threats, enabling further analysis of possible reasons for unsafe behavior and definition of safe composition specification (either cutting all unsafe input sequences, or redefining their responses in a safe way) for further component substitution via equation solving (Chapter 6).

Definition 4.16. Given the component TFSMs $\mathcal{S}$ and $\mathcal{P}$, the TFSM $\mathcal{C}$ with special designated output symbols *livelock* and *deadlock* and *lock* state is called the safety-aware parallel composition of $\mathcal{S}$ and $\mathcal{P}$, denote $\mathcal{C} = \mathcal{S} \Diamond_{safe} \mathcal{P}$, if its largest sub-machine without these special symbols and state is the parallel composition of given components, all transitions with *livelock* and *deadlock* outputs necessarily lead to the state *lock* and correspond to the safety threat in communication between $\mathcal{S}$ and $\mathcal{P}$. △

The Algorithm 4.3 describes how to derive such safety-aware composition.

After the safety-aware composition is derived, if the state *lock* is reachable and/or there are transitions with infinite sets of output delays, then at least one of the components should be substituted in order to provide a safe composition. For the purpose, locking transitions should be substituted with expected ones (or erased as forbidden) and the resulting TFSM could be used as a specification for the parallel equation.

The minimal value for the output delay on locking transitions provides a minimal value for a timed-bound, i.e., a minimal value for waiting for a response from the composition before deciding that (while using and/or testing a composite system) the system falls into deadlock or livelock and should be reset.

Algorithm 4.3 Safety-aware parallel composition

Input: The component TFSMs $\mathcal{S} = \langle S,\ I_S \cup V,\ O_S \cup U,\ s_0,\ \lambda_{\mathcal{S}},\ \Delta_{\mathcal{S}}, \sigma_{\mathcal{S}}\rangle$ and $\mathcal{P} = \langle P,\ I_P \cup U,\ O_P \cup V,\ p_0,\ \lambda_{\mathcal{P}},\ \Delta_{\mathcal{P}},\ \sigma_{\mathcal{P}}\rangle$

Output: The TFSM $\mathcal{C} = \langle C \cup \{lock\}, I_S \cup I_P, O_S \cup O_P \cup \{livelock,\ deadlock\}, c_0, \lambda_{\mathcal{C}}, \Delta_{\mathcal{C}}, \sigma_{\mathcal{C}}\rangle$ that is the parallel composition of given components augmented with safety parameters, $\mathcal{C} = \mathcal{S} \Diamond_{safe} \mathcal{P}$

1: Derive $Aut(\mathcal{S})$ and $Aut(\mathcal{P})$ using Algorithm 3.2.
2: Derive $Aut(\mathcal{S})_{\Uparrow I_P \cup O_P}$ by adding a loop in each state of $Aut(\mathcal{S})$ for all $a \in I_P \cup O_P$, derive $Aut(\mathcal{P})_{\Uparrow I_S \cup O_S}$ by adding a loop in each state of $Aut(\mathcal{P})$ for all $b \in I_S \cup O_S$.
3: Derive the global automaton $Globe = Aut(\mathcal{S})_{\Uparrow I_P \cup O_P} \cap Aut(\mathcal{P})_{\Uparrow I_S \cup O_S}$
4: Add the final state of *Globe* the special state $lock \in G_{fin}$.
Explore $Globe = \langle G,\ I_P \cup U \cup O_P \cup V,\ \delta G,\ \langle s_0, p_0\rangle, G_{fin}\rangle$: $G \subseteq S$ for deadlocks.
5: **for** each $\langle s, p\rangle \in G \setminus G_{fin}$ **do** where s is a state of $Aut(\mathcal{S})_{\Uparrow I_P \cup O_P}$ and p is a state of $Aut(\mathcal{P})_{\Uparrow I_S \cup O_S}$
6: **for** each $a \in U \cup V$ **do**
7: **if** $a \in actions_S(s) \setminus actions_P(p)$ or $a \in actions_P(p) \setminus actions_S(s)$ **then**
8: add to δ_G transition $\langle\langle s, p\rangle, deadlock, lock\rangle$
9: **end if**
10: **end for**
11: **end for**
Explore *Globe* for livelocks:
12: **for** each $\langle s, p\rangle \in G \setminus G_{fin}$ **do**
13: **if** $\langle s, p\rangle$ starts a cycle via non-final states only **then**
14: add to δ_G transition $\langle\langle s, p\rangle, livelock, lock\rangle$
15: **end if**
16: **end for**
17: Derive the intersection $\mathcal{G} = Globe \cap Aut(\mathcal{M}_{I, O \cup \{livelock, deadlock\}})_{\Uparrow U, V}$ ensuring that after each external input after some internal dialog the external output (or a decision on safety threat) is produced.
18: Derive the restriction to external alphabets $Aut(\mathcal{C}) = \mathcal{G}_{\Downarrow I \cup O \cup \{1\} \cup \{livelock, deadlock\}}$.
19: Restore $Aut(\mathcal{C})$ into TFSM $\mathcal{C}$ using the Algorithm 3.3.

In the next chapters, we assume that composition is always derived with safety-awareness, and hence denote the safety-aware composition operator simply by $\Diamond$.

4.5 Discussion

One of the interesting and perspective discussion about the model could be raised in generalization of the semantics of a timer associated with the TFSM. The crucial utility characteristics of timeouts, defined by means of the timer, is the possibility of the machine to produce different responses to the same requests depending on the value of the timer. Considering time as a resource, output delays as an amount of resource consumption per operation, the timeouts could be seen as boundary conditions, differentiating responses depending on the resource amount spent for applying the request. Therefore, changing interpretation of time, it is possible to use the same TFSM model to embed other quality parameters of service without changing the model.

For cumulative parameters, like time and cost, the defined correspondence with finite automata preserves its usefulness as defined, unrolling the value into a chain of transitions under one unit, as far as parameter value boundaries are integers. For other types of parameters (e.g., multiplicative like availability) another correspondence should be sought, or a transformation to another form of the metric should be done (for example, using logarithmic scale to switch from multiplication to addition).

A discussion about generalization of TFSM timer from representing time to any additive parameter is applicable while considering the composition, too. One of the essential correlation to be considered in TFSM parallel composition is the correlation of output delays of one component to timeouts in another. For example, changing from time to cost parameter, output delays become the price of the transition they associated with, and timeouts could be used to represent, for instance, different service levels depending on the amount spent on applied input. Then, the resulting output delays of the composition, accumulating all the hidden internal communications between components, corresponds to the overall price of the end-to-end operation. Output delays of one component causing timeout transitions in another are the prices of internal communications, and timeouts themselves represent decisions on the how to process each request depending on its price.

Though, the more thorough investigation of parameter-specific features of such TFSM is needed.

4.6 Chapter conclusions

In the chapter, we defined the parallel composition operator and investigated its properties as for deterministic as for nondeterministic and partial TFSM components. In communication of TFSM components both livelocks and deadlocks might appear, and, using catastrophic interpretation, we discuss these safety issues and provide algorithms for deriving safety-aware composition. Explicit consideration of time in the TFSM model allows to use time-bounded composition instead of length-bounded like in classical FSMs while restricting the livelock-dangerous internal communications between components. Cumulative nature of time parameter and established correspondence of TFSMs and FAs allows to efficiently derive parallel composition based on operations over finite automata preserving the correlation between output delays in one component to timeouts in another. Substituting time with other cumulative quality parameters while preserving model syntax and operators seems to be a perspective discussion for future work.

Chapter 5

Service Composition Optimization Based on TFSM Equation Solving

5.1 Introduction

Once the composition of the set of given services is derived and verified (Chapter 4), or during the monitoring of service usage (Chapter 2), it might turn out that one or several components are not working at their best or fail to provide the required composition, and hence should be either optimized or substituted.

Therefore, in this chapter, we address the following questions:

1. If the parallel composition of given components does not conform the specification, can it be fixed by substituting one of the components?
2. Given a set of requirements for composition and one of the components, whether it is possible to design another, somehow better component which in communication with the known one meets given composition requirements?
3. Given a set of requirements for composition and one of the components, whether it is possible to minimize the set with respect to another component (component under design)?

Within the TFSM framework, these questions could be answered based on solving the corresponding TFSM equation.

First, we provide an algorithm on deriving the largest (general) solution to the parallel TFSM equation with respect to reduction and equivalence conformance relations, based on solving the corresponding language equation from which the largest TFSM-language solution is extracted. Then, we discuss the special case of partial specification for composition, and two main approaches for solving the equation with respect to the quasi-conformance relations, based on completion of the specification and/or refining the complement operation. The latter is shown to be useful for the minimization of composition specification in cases when it is given as a set of requirements or tree FSM.

After the deriving the largest solution, it is crucial to choose solutions with required properties, and useful restricted solutions are discussed in the next chapter (Chapter 6).

5.2 TFSM parallel equation

Given a TFSM $\mathcal{A}$ with input alphabet $I_1 \cup U$ and output alphabet $V \cup O_1$, and TFSM $\mathcal{C}$ with input alphabet $I = I_1 \cup I_2$ and output alphabet $O = O_1 \cup O_2$. The expression

$$\mathcal{A} \Diamond \mathcal{X} \sim \mathcal{C}$$

is a parallel equation [71] over TFSMs, where TFSM $\mathcal{C}$ is the *specification* of the system while TFSM $\mathcal{A}$ describes the behavior of the known part of the system under design, usually called *context*, TFSM $\mathcal{X}$ is the *unknown component* to be designed with input alphabet $I_2 \cup V$ and output alphabet $U \cup O_2$, $\Diamond$ is the parallel composition operator and $\sim$ is the conformance relation required between the specification and composition.

Further, we consider the solution against the reduction ($\leq$) and the equivalence ($\cong$) relations between TFSMs, though, we show how the specification can be refined in order to treat quasi-reduction and quasi-equivalence in cases of partial specification.

Sometimes, the expression $\mathcal{A} \Diamond \mathcal{X} \leq \mathcal{C}$ is also called a parallel *inequality* over TFSMs.

The formal procedure to derive the largest solution for the TFSM equation, similar to classical FSMs, is based on solving the corresponding language equation, therefore, we briefly remind the known results for regular language equations [71].

5.2.1 Preliminaries on solving parallel language equations

As usual, $\mathcal{B}$ is a *solution* to the equation $\mathcal{A}\Diamond\mathcal{X} \sim \mathcal{C}$ if it holds $\mathcal{A}\Diamond\mathcal{B} \sim \mathcal{C}$.

An equation $\mathcal{A}\Diamond\mathcal{X} \sim \mathcal{C}$ is *solvable* if there exists a solution.

Solution $\mathcal{M}$ to the equation $\mathcal{A}\Diamond\mathcal{X} \sim \mathcal{C}$ is called the *largest solution* if it contains every solution to the equation.

Let the coefficients of the equation $\mathcal{A}\Diamond\mathcal{X} \sim \mathcal{C}$ be regular languages, the language $\mathcal{A}$ over alphabet $E_1 \cup E_2$, the unknown language $\mathcal{X}$ over alphabet $E_2 \cup E_3$, the language $\mathcal{C}$ over alphabet $E_1 \cup E_3$, the operation $\Diamond$ be the parallel composition and the conformance relation $\sim$ be the subset relation $\subseteq$.

As it is shown in [71], the largest solution to the language equation is given by the expression

$$Largest = \overline{\overline{\mathcal{C}}\Diamond\mathcal{A}} = \overline{(\overline{\mathcal{C}}_{\Uparrow E_2} \cap \mathcal{A}_{\Uparrow E_3})_{\Downarrow E_2 \cup E_3}}$$

If $Largest = \emptyset$ then the equation has only *trivial* solution[1].

If $\mathcal{A}\Diamond Largest = \mathcal{C}$ then the equation with respect to the equivalence relation is solvable. Otherwise, if $\mathcal{A}\Diamond Largest \subset \mathcal{C}$ then the equation with the equivalence relation has no solution and $\mathcal{C}' = \mathcal{A}\Diamond Largest$ is the largest composition language that can be implemented with the given context $\mathcal{A}$.

Solving equations over finite automata is a special case of solving equations over regular languages.

Given the equation $Aut(\mathcal{A})\Diamond Aut(\mathcal{X}) \leq Aut(\mathcal{C})$, its largest solution may be obtained by following procedure.

[1]By definition, for any languages A and C it holds $A\Diamond\emptyset = \emptyset$ and $\emptyset \subseteq C$, therefore, the $\emptyset$ is always a solution to parallel equation with reduction relation.

1. If the automaton $Aut(\mathcal{C})$ is nondeterministic and/or partial, then determinize it and complete (e.g. using demonic completion procedure). Make all the final states non-final and all the non-final states to be final, to obtain the automaton that accepts the complement of the language accepted by $Aut(\mathcal{C})$, denote $\overline{Aut(\mathcal{C})}$.

2. Build an automaton $\overline{Aut(\mathcal{C})}_{\Uparrow E_2}$ by adding a loop (s, e, s) for each state and each symbol $e \in E_2$.

3. Build an automaton $Aut(\mathcal{A})_{\Uparrow E_3}$ by adding a loop (s, e, s) for each state and each symbol $e \in E_3$.

4. Derive an intersection $\overline{Aut(\mathcal{C})}_{\Uparrow E_2} \cap Aut(\mathcal{A})_{\Uparrow E_3}$.

5. Restrict the derived intersection to the alphabets of unknown component, i.e. $(\overline{Aut(\mathcal{C})}_{\Uparrow E_2} \cap Aut(\mathcal{A})_{\Uparrow E_3})_{\Downarrow E_1 \cup E_2}$ by substituting each transition (s_1, a, s_2) with $a \notin E_1 \cup E_2$ with transition (s_1, ϵ, s_2) and further determinisation of the finite automaton with ϵ-transitions.

6. Derive the complement to the obtained automaton by interchanging final and non-final states, denote $Aut(\mathcal{S}) = \overline{(\overline{Aut(\mathcal{C})}_{\Uparrow E_2} \cap Aut(\mathcal{A})_{\Uparrow E_3})_{\Downarrow E_1 \cup E_2}}$. If $Aut(\mathcal{S})$ accepts non-empty language, then it is the largest automaton solution, otherwise the equation has only a trivial solution.

5.2.2 Deriving the largest solution for TFSM equation

Consider the equation $\mathcal{A} \Diamond \mathcal{X} \leq \mathcal{C}$, where coefficients $\mathcal{A}$ and $\mathcal{C}$ and the unknown component $\mathcal{X}$ are TFSMs, $\Diamond$ is TFSM parallel composition operator and $\leq$ is the reduction relation.

Since we established the correspondence between TFSMs and Finite Automata (Section 3.4.2), the TFSM equation similar to classical FSM equations [71] can be transformed and solved based on corresponding equation over regular languages:

$$\mathcal{A} \Diamond \mathcal{X} \leq \mathcal{C}$$

$$\Downarrow$$

$$Aut(\mathcal{A}) \Diamond Aut(\mathcal{X}) \cap Aut(\mathcal{M}_{\mathcal{C}}) \leq Aut(\mathcal{C})$$

$$\Downarrow$$

$$L(\mathcal{A})\Diamond L(\mathcal{X}) \cap L(\mathcal{M}_\mathcal{C}) \subseteq L(\mathcal{C})$$

which is using the known set theory transformation $A \cap B = C \Leftrightarrow A = C \cup \overline{B}$ is turned into

$$L(\mathcal{A})\Diamond L(\mathcal{X}) \subseteq L(\mathcal{C}) \cup \overline{L(\mathcal{M}_\mathcal{C})}.$$

The largest solution of the latter is known (Section 5.2.1) to be

$$L_{Solution} = \overline{L(\mathcal{A})\Diamond(\overline{L(\mathcal{C})} \cap L(\mathcal{M}_\mathcal{C}))}.$$

The language $L_{Solution}$ is regular, but in general case is not a TFSM language. The resulting language has to be intersected with the language of the maximal TFSM $\mathcal{M}_\mathcal{X}$ for the unknown component and then the largest prefix-closed subset of $L_{Solution} \cap L(\mathcal{M}_\mathcal{X})$ can be derived to obtain the largest TFSM solution. For short, denote I_X and O_X the input and the output alphabets of the unknown component correspondingly.

Proposition 5.1. *[79] The largest solution of the TFSM equation (inequality) $\mathcal{A}\Diamond\mathcal{X} \leq \mathcal{C}$ is the TFSM $\mathcal{S}$ such that $L(\mathcal{S})$ is the largest $[1^* \cup 1^* I_X 1^* O_X]$-prefix-closed subset of the language $L_{Solution} \cap L(\mathcal{M}_\mathcal{X})$, where $L_{Solution} = \overline{L(\mathcal{A}\Diamond(\overline{L(\mathcal{C})} \cap L(\mathcal{M}_\mathcal{C}))}$ and $\mathcal{M}_\mathcal{X}$ is the maximal TFSM for the unknown component. If $L_{Solution} = \emptyset$, then the equation has no solution. If $L_{Solution} \neq \emptyset$, but $L(\mathcal{S}) = \emptyset$, then the equation has no TFSM solution. If $L(\mathcal{S}) = 1^*$ then the equation (inequality) has only trivial solution.*

Example 5.1. *We illustrate the process of solving TFSM equation on a simplified Delivery Service, which specification is shown in Fig. 5.1(top). The specification provides the single quality and functional requirement: after the order is placed (request input symbol), the delivery should be made within 7 days.*

This Delivery Service consists of two components: Store service (shown in Fig. 5.1(bottom)) and Warehouse service, the latter to be designed.

The Store Service proceeds as follows: after the receiving of the order (request input), it checks the availability of the ordered item in the Warehouse (output check) within

one or two days. If the item is available (product input from Warehouse), it is sent for delivery to the customer (deliver output), otherwise (no_product input) the shipment from another stock is ordered and processed within two days (ship output), and after receiving the item (get_product) the checking of its compliance to the order is performed (check).

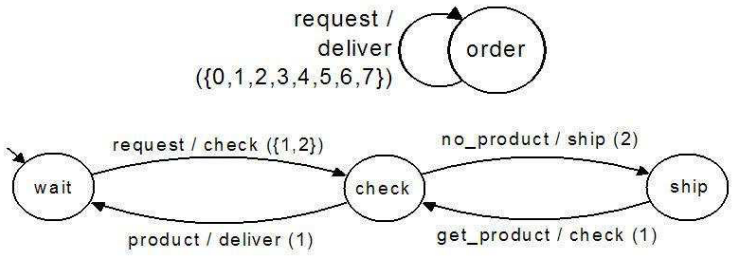

FIGURE 5.1: The Delivery Service composition specification (top) and the component service - Store Service (bottom)

The largest solution for the Warehouse component in such case is shown on Fig. 5.2.

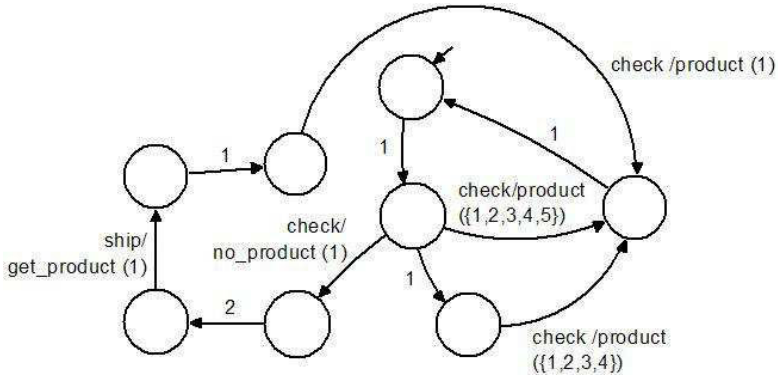

FIGURE 5.2: The largest solution for Warehouse component with restriction in delivery time

△

5.2.3 Verification of composition realizability

If the non-trivial largest solution exists, the next step is to check:

1. What is the maximal composition behavior that can be possible realized with a given context. That is, if $\mathcal{S}$ is the largest solution to the equation $\mathcal{X} \lozenge \mathcal{A} \leq \mathcal{C}$, then the maximal realizable composition is $\mathcal{C}' = \mathcal{S} \lozenge \mathcal{A}$ and the question is whether $\mathcal{C}' < \mathcal{C}$ or $\mathcal{C}' \cong \mathcal{C}$. The former means that the equation $\mathcal{X} \lozenge \mathcal{A} \cong \mathcal{C}$ has no solutions.

2. Whether the component under design/optimization can be implemented with required minimal behavior constrains without violating composition specification. I.e., given the requirement $\mathcal{X} \geq \mathcal{S}_{xmin}$, whether the derived largest solution satisfies it: $\mathcal{S} \geq \mathcal{S}_{xmin}$.

The latter question appears because of the web service composition interpretation we use. Since we do not consider possible involvement of the components in multi-user communications, the behavior of the component in the considered communication might be essentially nondeterministic.

Let us illustrate this intuition on the following example of simple delivery service.

Example 5.2. *Since there is no control from the store over the availability of the purchased product at the warehouse, it is essential that the warehouse service is able to response both "available"or "not available"(outputs product and no_product) when the the request "check"is made for the first time. Though, the largest solution in Fig. 5.2 to the equation with quality restriction "delivery within 7 days"in one of the states allows only response "available"(output product). It indicates that either the store service in Fig. 5.1 should be optimized or the quality restriction should be revised.* △

The minimal behavior restriction is formalized as TFSM $\mathcal{S}_{xmin}$ and is assumed to be consistent with composition specification, i.e.,

$$\mathcal{A} \diamondsuit \mathcal{S}_{xmin} \leq \mathcal{C}$$

then the TFSM equation solving is considered as solving the system of equations

$$\begin{cases} \mathcal{A} \diamondsuit \mathcal{X} \leq \mathcal{C} \\ \mathcal{X} \geq \mathcal{S}_{xmin} \end{cases}$$

which is obviously reduced to a single equation when the minimal component behavior is trivial TFSM, i.e. if $L(\mathcal{S}_{xmin}) = 1^*$.

If the solution to the given TFSM equation does not exist or does not satisfy the minimal behavior requirement, it is necessary to check whether the problem occurs due to functional or non-functional requirements. It can be done by checking if the equation is solvable when the specification has no delay restrictions.

If the equation with no delay restrictions is solvable, then it means that the initial quality requirements cannot be met by the given components and either the context requires optimization or requirements should be revised. If the equation with no output delay restrictions is unsolvable, then it means that the required functionality cannot be implemented with the given component whatever other component it would be communicating with.

Example 5.3. *In Example 5.2, the retrieval point service cannot be implemented with required nondeterminism to guarantee the delivery within 7 days. Therefore, we attempt to derive the largest solution for this service without restrictions on the delivery delay. The largest solution for this case is shown in Fig 5.3 and does satisfy the requirement for supporting nondeterministic response for "check"request. Transitions denoted with * correspond to don't care behavior of the component.*

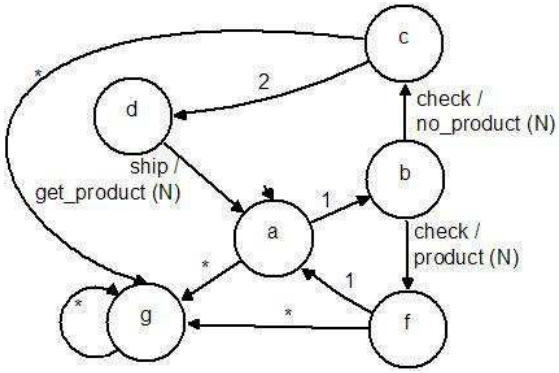

FIGURE 5.3: The largest solution for Warehouse component with no time restrictions in composition specification

△

5.3 Solving equations for partial specifications

Often, the service composition specification is given as a set of requirements. Therefore, it is rational to consider not equivalence and/or reduction relations but rather quasi-conformance relations for partial machines while solving corresponding equation, i.e., in this section we consider the equation

$$\mathcal{X} \Diamond \mathcal{A} \lesssim \mathcal{C}$$

One approach is to complete the partial TFSM $\mathcal{C}$ based on the considered interpretation of specification partiality, and transform the equation to

$$\mathcal{X} \Diamond \mathcal{A} \leq \mathcal{C}_{completed}$$

.

By definition, $in_{\mathcal{C}}(c_0)$, where c_0 is the initial state of the TFSM $\mathcal{C}$, denotes the set of all (timed) input sequences for which the behavior of $\mathcal{C}$ is defined.

Then, the set of input sequences, for which the behavior of $\mathcal{C}$ is undefined, is the set $undef_{\mathcal{C}} = (I \times \mathbb{N})^* \setminus in_{\mathcal{C}}(c_0))$.

We distinguish two types of undefined input sequences in partial specification: forbidden and "don't care"ones. Let us denote $forbid_{\mathcal{C}} \subseteq undef_{\mathcal{C}}$ the set of the input sequences that are forbidden for the composition, while $indif_{\mathcal{C}} \subseteq undef_{\mathcal{C}}$ being the set of "don't care"input sequences that allow any behavior the composition. For consistency reasons, it is assumed $forbid_{\mathcal{C}} \cap indif_{\mathcal{C}} = \emptyset$ and $forbid_{\mathcal{C}} \cup indif_{\mathcal{C}} = undef_{\mathcal{C}}$.

Then the completion of the partial specification is made as follows:

$$trace_{\mathcal{C}_{completed}} = trace_{\mathcal{C}} \cup \{indif_{\mathcal{C}} \times (O \times \mathbb{N})^*\}.$$

If $forbid_{\mathcal{C}} \neq \emptyset$ then $\mathcal{C}_{completed}$ is still a partial TFSM, but all undefined input sequences are considered only as forbidden ones and therefore the quasi-reduction relation in the equation is consistently substituted with reduction relation.

Another approach is based on the key observation, that the chosen conformance relation for the equation can be taken into account by revising the complementation operation while deriving the largest solution.

Informally, the process of solving FSM equation may be summarized as following:

1. Derive the forbidden behavior of the composition. While considering strict equivalence and reduction relations or/and the specification is complete, this step corresponds to the complement operation over the corresponding specification automaton.

2. Derive the global behavior that is possible with the given context and induces the forbidden behavior of the composition. This step corresponds to the extension of the automaton representing forbidden composite behavior to internal actions and intersection with the automaton representing the behavior of the context.

3. Extract the behavior of the component under optimization (unknown component) that may cause the forbidden behavior of the composition. Using automata representation, this step corresponds to the restriction operation applied to the automaton derived at the previous step.

4. Derive the allowed behavior of the unknown component, i.e., the largest behavior of the unknown component that in communication with the context does not cause the forbidden behavior of composition. Using automata representation, this step corresponds to the complement operation applied to the automaton derived at the previous step.

This generalized idea behind the FSM equation solving can be effectively adapted for solving equations with partial specifications and quasi- conformance relations by refining the complement operation instead of specification completion.

If applied to the set of requirements, analysis of the results of the Step 3 of described process can be used for minimizing the set of compositional requirements with respect to the unknown component (component under optimization) as we show in the Section 5.4.

Consider the TFSM $\mathcal{C}$ with alphabets I and O, and the language $L(\mathcal{C})$ of its corresponding finite automata. Derive the maximal TFSM $\mathcal{P}_C$ defined over the same input sequences, i.e., the TFSM with the same set of input sequences but with any output response, i.e., $trace_{\mathcal{P}_C} = in_{\mathcal{C}}(c_0) \times (O \times \mathbb{N})^*$. Then, we call the *partial complement* for $L(\mathcal{C})$ the language $\overline{L(\mathcal{C})}^p = L(\mathcal{P}_C) \setminus L(\mathcal{C})$.

The largest solution of the TFSM equation $\mathcal{A} \Diamond \mathcal{X} \lesssim \mathcal{C}$ is the TFSM $\mathcal{S}$ such that $L(\mathcal{S})$ is the largest $[1^* \cup 1^* I_X 1^* O_X]$-prefix-closed subset of the language $L_{Solution} \cap L(\mathcal{M}_{\mathcal{X}})$, where $L_{Solution} = \overline{L(\mathcal{A} \Diamond (\overline{L(\mathcal{C})}^p))}^p$ and $\mathcal{M}_{\mathcal{X}}$ is the maximal TFSM for the unknown component. If $L_{Solution} = \emptyset$, then the equation has no solution. If $L_{Solution} \neq \emptyset$, but $L(\mathcal{S}) = \emptyset$, then the equation has no TFSM solution. If $L(\mathcal{S}) = 1^*$ then the equation has only trivial solution.

5.4 Minimizing composition specification w.r.t. component under optimization

Given the composite service specification as a set of requirement, the question is whether it can be minimized preserving the requirements applied for the component under optimization.

In this sections, we consider the component under optimization to be *embedded*, i.e., having empty external alphabets and communicating with the environment only via the context component.

5.4.1 Determining the set of all internal traces violating a given composition requirement

In this section, given the TFSMs $\mathcal{C}$ and $\mathcal{B}$, the TFSM $\mathcal{B}$ being embedded, a timed external input sequence α and the expected output response β of the TFSM $\mathcal{C}\Diamond\mathcal{B}$ to α, a procedure is proposed for deriving a special automaton $R(\alpha\beta)$ over internal alphabets such that an internal trace takes the automaton to a final state if and only if this trace violates the given composition requirement $\alpha\beta$. We can then check which external traces are violated by the same sets of internal traces.

Example 5.4. *The minimization of the set of requirements we consider with an abstract TFSM example, for more illustrative purpose.*

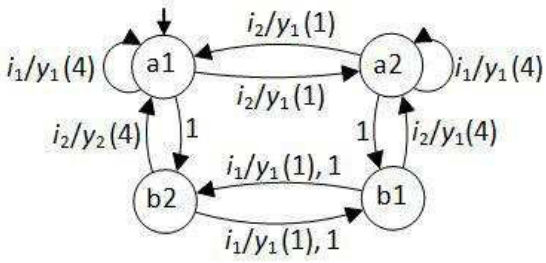

FIGURE 5.4: Composite TFSM

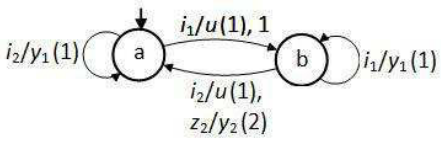

FIGURE 5.5: The context TFSM

Consider the composed TFSM in Fig. 5.4 and the context TFSM in Fig. 5.5. The following set of requirements TS can be applied for the composition:

$TC_1 = \langle i_1, 1\rangle.\langle i_2, 1\rangle/\langle y_1, 1\rangle.\langle y_2, 4\rangle$

$TC_2 = \langle i_1, 1\rangle.\langle i_1, 0\rangle.\langle i_2, 0\rangle/\langle y_1, 1\rangle.\langle y_1, 1\rangle.\langle y_1, 4\rangle$

$TC_3 = \langle i_1, 1\rangle.\langle i_2, 0\rangle.\langle i_1, 0\rangle/\langle y_1, 1\rangle.\langle y_1, 4\rangle.\langle y_1, 4\rangle$

$TC_4 = \langle i_2, 1\rangle.\langle i_1, 0\rangle/\langle y_2, 4\rangle.\langle y_1, 4\rangle$

$TC_5 = \langle i_1, 0\rangle.\langle i_1, 0\rangle/\langle y_1, 4\rangle.\langle y_1, 4\rangle$

$TC_6 = \langle i_2, 0\rangle.\langle i_2, 1\rangle/\langle y_1, 1\rangle.\langle y_1, 4\rangle$

$TC_7 = \langle i_2, 0\rangle.\langle i_1, 1\rangle/\langle y_1, 1\rangle.\langle y_1, 1\rangle$

$TC_8 = \langle i_2, 0\rangle.\langle i_1, 0\rangle.\langle i_1, 0\rangle/\langle y_1, 1\rangle.\langle y_1, 4\rangle.\langle y_1, 4\rangle$

$TC_9 = \langle i_2, 0\rangle.\langle i_2, 0\rangle.\langle i_1, 0\rangle/\langle y_1, 1\rangle.\langle y_1, 1\rangle.\langle y_1, 4\rangle$

Respectively, a global trace induced by the input sequence $\langle i_1, 1\rangle.\langle i_2, 1\rangle$ *can be:* $a1{-}1 \to b1{-}i_1 \to m1{-}1 \to n1{-}y_1 \to b1{-}1 \to b1{-}i_2 \to t1{-}1 \to r1{-}u \to a2{-}1 \to b3{-}z_2 \to q1{-}1 \to c1{-}1 \to d1{-}y_1$.

Since $TC_1 = \langle i_1, 1\rangle.\langle i_2, 1\rangle/\langle y_1, 1\rangle.\langle y_2, 4\rangle$ *we conclude that if an embedded component has an internal trace* $\langle u, 4\rangle.\langle z_1, 1\rangle$ *then such implementation of the embedded component TFSM* $\mathcal{B}$ *violates the requirement* TC_1.

On the other hand, a global trace corresponding to $\langle i_1, 1\rangle.\langle i_1, 0\rangle.\langle i_2, 0\rangle$ *can be:* $a1{-}1 \to b1{-}i_1 \to m1{-}1 \to n1\ {-}y_1 \to b1{-}i_1 \to m1{-}1 \to n1{-}y_1 \to b1{-}i_2 \to t1{-}1 \to r1\ {-}u \to a2{-}1 \to b3\ {-}z_1 \to p1{-}1 \to h1{-}1 \to s1{-}y_2$.

Since $TC_2 = \langle i_1, 1\rangle.\langle i_1, 0\rangle.\langle i_2, 0\rangle/\langle y_1, 1\rangle.\langle y_1, 1\rangle.\langle y_1, 4\rangle$ *we conclude that if an embedded component has an internal trace* $\langle u, 4\rangle.\ \langle z_1, 1\rangle$ *then such implementation of the embedded component TFSM* $\mathcal{B}$ *violates the requirement* TC_2. △

Given the composition requirement $\alpha\beta$, according to the structure of an implementation system, all the possible global traces of a composition under design are traces of the given context. Correspondingly, we can leave in the context TFSM only those global

traces that are induced by the external input sequence α (steps 1-3 in Algorithm 5.1). First, we derive the automaton $A(\alpha\beta)$ accepting the only trace, corresponding to the given composition requirement. Second, we derive the automaton $D(\alpha\beta) = M_\alpha \setminus A(\alpha\beta)$ that contains all possible external traces with output sequences different from the reference (given in the requirement) output sequence. Then, at steps 4 and 5 in Algorithm 5.1, we extract from the context automaton the submachine that has the set of traces $\{\nu|\nu_{\Downarrow I\cup O\cup\{1\}} \in D(\alpha\beta)\}$, i.e., those global traces that correspond to an unexpected external output sequences when a timed input sequence α is applied to the context TFSM. Last (steps 6-8 in Algorithm 5.1), we hide external symbols in the intersection and obtain the set of internal traces that can be traversed in the composition when the output sequence of the composed TFSM to α is different from β. The sequence of transitions under 1 after the last internal output $v \in V$ does not influence violating internal traces and thus, it may be eliminated.

Algorithm 5.1 Deriving a set of internal traces violating a composition requirement $\alpha\beta$.

Input: The context TFSM $\mathcal{C}$ and an external composition requirement trace $\alpha\beta = \langle i_1, t_1\rangle.\langle o_1, k_1\rangle.\langle i_2, t_2\rangle.\langle o_2, k_2\rangle \dots \langle i_n, t_n\rangle.\langle o_n, k_n\rangle$.

Output: An automaton $R(\alpha\beta)$ accepting all internal traces that lead to violation of the requirement $\alpha\beta$.

Deriving violating external traces with the input sequence α:

1: Derive an automaton $A(\alpha\beta)$ with the initial state s_0 that accepts the only sequence $1^{t_1}i_1 1^{k_1} o_1 \dots 1^{t_n} i_n 1^{k_n} o_n$.

2: Derive an automaton M_α with the set $1^{t_1}i_1 1^* O \dots 1^{t_n} i_n 1^* O$ of traces.

3: An automaton $B(\alpha\beta) = M_\alpha \setminus A(\alpha\beta)$ accepts all violating external traces with the input sequence α.

Deriving global traces with the external input sequence α and external output sequence different from β:

4: Derive the automaton $B(\alpha\beta)_{\Uparrow U\cup V}$, i.e., expand the automaton $B(\alpha\beta)$ to the internal alphabets adding loops at each state labeled by internal symbols.

5: Intersect $B(\alpha\beta)_{\Uparrow U\cup V}$ with the automaton of the context TFSM $\mathcal{C}$: $G(\alpha\beta) = B(\alpha\beta)_{\Uparrow U\cup V} \cap Aut(\mathcal{C})$.

Deriving the set of violating internal traces:

6: Hide in $G(\alpha\beta)$ external symbols, and replace each sequence of transitions $v(1)^*$, $v \in V$, to a final state with a transition labeled by v.

7: Denote $G'(\alpha\beta)_{\Downarrow U\cup V\cup\{1\}}$ the obtained automaton.

8: Intersect $G'(\alpha\beta)_{\Downarrow U\cup V\cup\{1\}}$ with the automaton $Aut(U, V)$: $R(\alpha\beta) = G'(\alpha\beta)_{\Downarrow U\cup V\cup\{1\}} \cap Aut(U, V)$.

Proposition 5.2. *Given the specification requirement $\alpha\beta$ of the composed TFSM $\mathcal{C}\Diamond\mathcal{B}$ and the automaton $R(\alpha\beta)$, returned by the Algorithm 5.1, an implementation $\mathcal{E}$ of the*

embedded component $\mathcal{B}$ is violating the requirement $\alpha\beta$ if and only if $\mathcal{E}$ has a trace that takes the automaton $R(\alpha/\beta)$ from the initial state to a final state.

Proof. The requirement $\alpha\beta$ is violated by an implementation $\mathcal{E}$ of $\mathcal{B}$ if and only if it is violated by some internal trace γ that is a trace of $\mathcal{E}$. It means that there exists a global trace $\nu \in Globe(\mathcal{C}\Diamond\mathcal{E})$ such that $\nu_{\Downarrow I\cup O\cup\{1\}} = \alpha\beta'$ where $\beta' \neq \beta$ and $\nu_{\Downarrow U\cup V\cup\{1\}} \in \gamma 1^*$. By construction, $\alpha\beta$ is accepted by $A(\alpha\beta)$ and for all $\beta' \neq \beta$ it holds $\alpha\beta'$ is a trace of $D(\alpha\beta)$, i.e. $\nu_{\Downarrow I\cup O\cup\{1\}}$ is a trace of $D(\alpha\beta)$, therefore, due to the properties of the restriction and expansion operations, all the traces $\nu \in (\nu_{\Downarrow I\cup O\cup\{1\}})_{\Uparrow U\cup V}$ are accepted by $D(\alpha\beta)_{\Uparrow U\cup V}$. From the definition of the global automaton, ν is accepted by $Aut(\mathcal{C})$, then ν is a trace of $Aut(\mathcal{C}) \cap D(\alpha\beta)_{\Uparrow U\cup V} = G(\alpha\beta)$ and $\nu_{\Downarrow U\cup V\cup\{1\}}$ is accepted by $G(\alpha\beta)_{\Downarrow U\cup V\cup\{1\}}$. Since $\nu_{\Downarrow U\cup V\cup\{1\}} \in \gamma 1^*$ and to derive $G'(\alpha\beta)_{\Downarrow U\cup V\cup\{1\}}$ we cut off all the suffixes containing symbol 1 only, we conclude that γ is accepted by $G'(\alpha\beta)_{\Downarrow U\cup V\cup\{1\}}$ and hence γ takes the $R(\alpha\beta)$ from initial state to a final state. □

5.4.2 Minimizing the set of requirements

The minimization of the set of requirements allows to make the testing and verification process faster after the substitution of a component is made, because the complexity of checking whether composition meets the requirements or not is essentially depending on the number of requirements to check.

The minimization of the set of requirements can be made based on the similar process as solving an equation.

The problem of the set of requirements minimization is stated as follows: given a composition specification in form of the set of input/output sequences (the set of requirements) TS and the context $\mathcal{A}$, we are required to derive a minimal subset $TS' \subseteq TS$ such that the composition $\mathcal{A}\Diamond\mathcal{X}$ satisfy TS' if and only if $\mathcal{A}\Diamond\mathcal{X}$ satisfy TS.

In other words, if the set of requirements TS is represented by the TFSM $\mathcal{C}_{TS}$, then we are required to derive such $\mathcal{C}_{TS'} \leq \mathcal{C}_{TS}$ so that the equations $\mathcal{X}\Diamond\mathcal{A} \lesssim \mathcal{C}_{TS}$ and $\mathcal{X}\Diamond\mathcal{A} \lesssim \mathcal{C}_{TS'}$ have the same sets of solutions.

Similar to the test minimization procedures for classical FSMs [80], the minimization of the set of requirement for composition includes the following steps:

1. Delete from the set of requirements TS requirements that are applied to the context only and do not induce any communication between components.

2. Determine for each requirement in TS the set of internal traces that violate the requirement, derive the union R of such internal traces and determine a subset $TS' \subseteq TS$ of external requirements such that the union of the sets of traces violating requirements TS' coincides with R.

In this case, the full set of requirements TS is met if and only if TS' is.

Example 5.5. *For the running example, consider requirements* $TC_1 = \langle i_1, 1\rangle.\langle i_2, 1\rangle/\langle y_1, 1\rangle.\langle y_2, 4\rangle$ *and* $TC_2 = \langle i_1, 1\rangle.\langle i_1, 0\rangle.\langle i_2, 1\rangle/\langle y_1, 1\rangle.\langle y_1, 1\rangle.\langle y_2, 4\rangle$. *The set of unexpected external traces for* $\alpha_1\beta_1$ *is* $\{\langle i_1, 1\rangle.\langle o_1, k_1\rangle.\langle i_2, 1\rangle.\langle o_2, k_2\rangle \mid o_1, o_2 \in Y \wedge k_1, k_2 \in \mathbb{N}, (o_1 \neq y_1 \vee k_1 \neq 1 \vee o_2 \neq y_2 \vee k_2 \neq 4)\}$ *and the automaton* $D(\alpha_1\beta_1)$ *with this set of traces is represented in Fig. 5.6 where actions corresponding to the timed input sequence* α_1 *are shown in bold. The reference behavior is represented with a sequence of transitions* $e_0 - 1 \to e_1 - i_1 \to e_2 - 1 \to e_3 - y_1 \to e_4 - 1 \to e_5 - i_2 \to e_6 - 1 \to e_7 - 1 \to e_8 - 1 \to e_9 - 1 \to e_10 - y_1$, *and therefore this sequence is eliminated from automaton* $D(\alpha_1\beta_1)$.

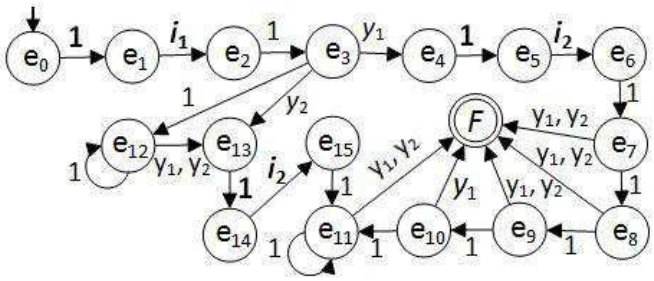

FIGURE 5.6: The automaton $D(\alpha_1\beta_1)$ with the set of unexpected external traces for requirement TC_1

At the next step, we add loops under internal actions u, z_1, *and* z_2 *at each state of the automaton* $D(\alpha_1\beta_1)$ *and then intersect the obtained automaton with the automaton* $Aut(\mathcal{C})$ *for the context TFSM. At the last step we hide all external symbols* i_1, i_2, y_1 *and* y_2 *and the obtained automaton* $R(\alpha_1\beta_1)$ *is shown in Fig. 5.7, where there is the only accepting state* F *(Fail). According to the above procedure, states* 9 *and* 10 *will be merged with the state* F, *and thus, a requirement* TC_1 *is violated by each component TFSM* $\mathcal{B}$ *that has a trace from the set* $\{\langle u, 4\rangle.\langle z_2, k\rangle, k \in \mathbb{N}\} \cup \{\langle u, 4\rangle.\langle z_1, k\rangle, k \in \mathbb{N} \setminus \{1\}\}$ *and only such implementations.*

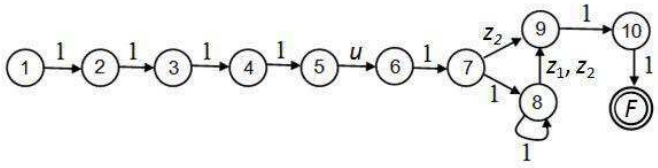

FIGURE 5.7: The set of internal traces violating the requirement TC_1

After deriving $R(\alpha_2\beta_2)$ for the TC_2, one can check that $R(\alpha_1\beta_1)$ and $R(\alpha_2\beta_2)$ have the same languages, i.e. TC_1 and TC_2 are violated by the same set of faulty implementations of the embedded component TFSM $\mathcal{B}$ and, hence, TC_1 is satisfied if and only if TC_2 is satisfied, and one of them therefore can be deleted from the set of requirements. The input sequence of TC_2 is longer and thus, TC_2 is preferable to be deleted. In the same way, one can assure that requirements TC_6 and TC_3 are violated by one and the same set of internal traces and therefore, TC_6 also can be deleted. As a result, six of requirements are sufficient to satisfy while substituting the embedded component, i.e. a minimized set of requirements $TS_{min} = \{TC_1, TC_3, TC_4, TC_5, TC_8, TC_9\}$ can be used while optimizing the embedded component. △

Example 5.6. *For the Loan Approval Service from Example 3.4, the set $TS = \{\langle IAr, k\rangle.\langle OK, t\rangle/\langle A, 8\rangle.\langle out1, 1\rangle : \forall k \geq 0, t \leq 10; \langle IAr, k\rangle/\langle NA, 8\rangle : \forall k \geq 0\}$ of given requirements can be minimized down to $TS_{min} = \{\langle IAr, 1\rangle.\langle OK, 1\rangle/\langle A, 8\rangle.\langle out1, 1\rangle; \langle IAr, 1\rangle/\langle NA, 8\rangle\}$.*

△

5.5 Chapter conclusions

We proposed how to derive the largest solution, i.e., the collection of all component service behaviors that can substitute given (unknown) component under optimization. The further choice of restricted solutions with required properties is discussed in Chapter 6.

Also, based on the equation solving procedure, we showed how the set of composition requirements, given in the form of input-output sequences (corresponding as to a list of requirements and/or a collection of desired scenarios to fulfill), can be minimized with respect to the component under optimization. The minimization of requirement set can be used, for example, for fastening the testing and verification of the composition against given set of requirements after each substitution of the component under optimization.

Chapter 6

Extracting Restricted Solutions with Required Properties

6.1 Introduction

In this section we consider some intuitions on how useful and interesting particular solutions might be extracted from the largest solution of the solvable equation.[1]

Primarily, we consider algorithms of deriving sub-machines of the largest solution rather than reductions, since similar to classical FSMs, the task of deriving a reduction with desirable properties is known to be quite complicated [71], while for a sub-machine extraction efficient algorithms might be developed.

Since the main objectives of the thesis are concerned on web services optimization, we extract restricted solutions interesting from the service optimization points of view.

In particular, we consider the issues of extracting safe components with manageable quality properties.

Therefore, the following restricted solutions are considered in this section:

1) *faster component*: at each state for each input the transition with the smallest output delay value is chosen. This restricted solution being composed with the given context

[1]The results of this chapter have been partly obtained within the project №14-08-31640 mol_a "Software quality evaluation based on logic circuit analysis and learning"funded by Russian Foundation for Basic Research (project coordinator - Natalia Kushik)

allows to estimate achievable quality of the composition with the given context, and in case it is not satisfactory, either reconsider the specification promises or redesign the context; though, the existence of faster reductions, rather than sub-machines, might be further considered, e.g., by unrolling cycles before choosing a sub-machine;

2) *least restrictive requirements*: choosing the slowest transitions in the component under design, we evaluate the most relaxed requirements the component needs to possess in order to have the composition that still conforms to the specification;

3) *maximizing/minimizing timeouts in the component*: the fact, that the component is not reachable from the context at any time instance with any possible input allows to manipulate timeout values for some states. Deriving the component with maximized timeouts might be useful to make the component more tolerant towards the quality violations in the context, preserving the same functionality during longer time intervals. On the other hand, for more time-sensitive services, it might be useful to minimize timeouts instead, correlating them to the session timeouts and producing appropriate error or reset messages if the context component violates its quality requirements.

4) *safe components*: choose those restricted solutions that do not cause any livelocks or deadlocks in composition with the given context.

5) *minimal changes required*: minimize distance in the number of transitions with the initial component that was decided to be optimized/substituted, e.g., because of safety reasons. This can be done by some special intersection of the largest solution with this initial component, assuming, that if the largest solution has variants of transition options for a transition of the initial component, then the latter is chosen, otherwise, the transition between variants is chosen based on output delay value. The choice from multiple options for the output responses in the largest solution that are all differ from the ones of the initial component might be done by any convenient ranking of the outputs. For example, based on semantical interpretation of abstract outputs, or by preferences of one type of messages over others.

6.2 Safe component selection

In this section we show that the largest livelock-safe solution is exactly the one obtained for the equation with restricted values for output delay.

Livelock-safe solutions are reductions (sub-machines in simplified case) of the largest solution for the equation with the specification with restricted output delays values, in cases when the context has the minimal non-zero output delay.

Otherwise, the component should be chosen to have non-zero delays.

Unlike classical FSMs, the TFSMs do not require usage of l-bounded composition operator in equations in order to derive livelock-safe components, since, as we show in Section 4.4, if at least one of the components of composition has non-zero minimal output delay, then l-bounded composition could be substituted by time-bounded one. Therefore, it is enough to bound specification and use the general parallel composition operator.

However, the question of how to choose an appropriate time bound may arise. If the equation does not have non-trivial solutions with given bound, it does not mean that it does not have livelock-safe solutions with greater bound.

Example 6.1. *The largest solution for the warehouse from Example 5.3 shown in Fig. 5.3 does not guarantee livelock-safety, while the largest solution derived for the time-restricted specification in Fig. 5.2 does.* △

6.3 Most similar substitution

As noted before, the task of solving an equation may be applied for the component substitution problem, if the given components are unable to provide the required composition with desirable quality. In this case, the substituting component may be sought as some restricted solution to the corresponding equation.

In this section we discuss how to choose this restricted solution being similar to the initially given component, i.e. such solution that would require as least as possible changes to be made in initial component to satisfy the composition specification.

Therefore, we consider given service components $\mathcal{S}$ and $\mathcal{P}$ and the composition specification $\mathcal{C}$, so that $\mathcal{S}\Diamond\mathcal{P} \not\sim \mathcal{C}$, where $\sim \in \{\cong, \leq, \sqsupseteq, \lesssim\}$ is a particular conformance relation.

We assume that at least one of the components should be re-designed/optimized, which may be done via solving a corresponding equation $\mathcal{X}\Diamond\mathcal{P} \sim \mathcal{C}$. Using the results of Chapter 5, the largest solution $\mathcal{S}_X$ to this equation may be derived (for any chosen conformance relation).

The goal of this section is to extract all those solutions that preserve the 'correct'part of the behavior of the given component $\mathcal{S}$ and modify only those transitions that lead to the composition specification violations.

Since all the traces of the component under optimization that are not violating the specification are contained in the largest solution, then the largest correct part of the behavior of the given component is contained in the TFSM $\mathcal{S}_{conf} = \mathcal{S} \cap \mathcal{S}_X$. If $L(\mathcal{S}_{conf}) = 1^*$, i.e. $traces_{\mathcal{S}conf} = \{\epsilon\}$, then the component $\mathcal{S}$ should be re-designed completely and no parts of its behavior could be included into the solution.

For a given component $\mathcal{S} = \langle S, I_X, O_X, s_0, \lambda_{\mathcal{S}}, \Delta_{\mathcal{S}}, \sigma_{\mathcal{S}}\rangle$ and the derived largest solution $\mathcal{S}_X = \langle X, I_X, O_X, x_0, \lambda_{\mathcal{X}}, \Delta_{\mathcal{X}}, \sigma_{\mathcal{X}}\rangle$, the algorithm is based on three main steps:

1. Determine, which part of the initially given component induces the behavior conformant to the specification composition, and preserve it, i.e., preserve in the solution all traces of the initially given component that are included in the largest solution $(traces_S(s_0) \cap traces_X(x_0))$.

2. For all input sequences that are undefined in the initially given component, choose output responses from the largest solution, i.e., add to the solution traces $\{\alpha\beta | \alpha \notin in_S(s_0) \wedge \alpha\beta \in traces_X(x_0)\}$.

3. For the part of the given component that violates composition specification, i.e., for the traces from the set $traces_S(s_0) \setminus traces_X(x_0)$, re-define output responses from the largest solution, i.e., add to the solution traces $\{\alpha\beta | \alpha \in in_S(s_0) \wedge out_S(s_0, \alpha) \cap out_X(x_0, \alpha) = \emptyset \wedge \alpha\beta \in traces_X(x_0)\}$.

Constructively, the extracting of the solution close to the initial component, is based in the intersection of the given component and the largest solution, with some minor modifications to take into account steps 2 and 3.

Algorithm 6.1 considers the reduction conformance relation. As we shown in Chapter 5, quasi-reduction relation in equation solving can be transformed to reduction by redefining the specification for don't care sequences, while equivalence and quasi-equivalence relations just require additional verification of the result. Therefore, Algorithm 6.1 may be easily adapted to treat other than reduction relation if necessary.

Proposition 6.1. *The TFSM $\mathcal{R}$ derived by the Algorithm 6.1 is a solution to the equation $\mathcal{X}\Diamond\mathcal{P} \leq \mathcal{C}$.*

Proof. By construction, $L(\mathcal{R}) \subseteq L(\mathcal{S}_X)$, i.e., $\mathcal{R}$ is a reduction to $\mathcal{S}_X$ and hence $\mathcal{R}\Diamond\mathcal{P} \leq \mathcal{S}_X\Diamond\mathcal{P} \leq \mathcal{C}$. □

Remark 6.2. While comparing the outputs of the given component to the outputs allowed in the largest solution, when outputs of the initially given component should be substituted (at the step 16), the decision on the choice of one output option from the largest solution might be done based on some similarity function defined over output symbols. In particular, in cases when the TFSM representation of the service components behavior is derived from the data-enabling models (like TEFSM [23] or STS [81, 82]), the values of the data variables are split into domains and those domains are encapsulated into new abstract input and output symbols. Then, new output symbols corresponding to the same output with different data values might be considered as more similar to each other than the symbols corresponding to different data values, and the preference in output symbols substitution might be based on such similarity.

6.4 Managing output delays values

Two approached can be used to restrict the output delays for extracting solutions from the largest solution.

1. First, the solutions with upper and lower bounds on output delays value might be considered.

   Given the largest solution $\mathcal{S}_X$ with alphabets I_X and O_X, the largest solution with output delays not less than k is the TFSM $\mathcal{S}_{X,\geq k}$ with the language $L(\mathcal{S}_{X,\geq k}) = L(\mathcal{S}_X) \cap (1^* I_X 1^k 1^* O_X)^* 1^*$.

Algorithm 6.1 Extracting the solution containing the correct behavior of the given component

Input: The equation $\mathcal{X}\Diamond\mathcal{P} \sim \mathcal{C}$, the largest solution $\mathcal{S}_X = \langle X, I_X, O_X, x_0, \lambda_\mathcal{X}, \Delta_\mathcal{X}, \sigma_\mathcal{X}\rangle$ and the component $\mathcal{S} = \langle S, I_X, O_X, s_0, \lambda_\mathcal{S}, \Delta_\mathcal{S}, \sigma_\mathcal{S}\rangle$ so that $\mathcal{S}\Diamond\mathcal{P} \not\sim \mathcal{C}$.

Output: The TFSM $\mathcal{R} = \langle R, I_X, O_X, r_0, \lambda_\mathcal{R}, \Delta_\mathcal{R}, \sigma_\mathcal{R}\rangle$ so that $\mathcal{R}\Diamond\mathcal{P} \sim \mathcal{C}$ and $\mathcal{R}$ is as similar as possible to $\mathcal{S}$.

Add to R the initial state is $r_0 = \langle\langle s_0, t_S = 0\rangle, \langle x_0, t_X = 0\rangle\rangle$.
Set current state $r = r_0$, mark r unexplored.
for each unexplored $r = \langle\langle s, t_S\rangle, \langle x, t_X\rangle\rangle \in R$ **do**
    for each $i \in I_X$ **do**
        if $i \in inputs_S(s) \cap inputs_X(x)$ **then**
            if $outputs_S(s,i) \cap outputs_X(x,i) \neq \emptyset$ **then**
                for each $o \in outputs_S(s,i) \cap outputs_X(x,i)$ **do**
                    $r' = \langle\langle s', 0\rangle, \langle x', 0\rangle\rangle$, where $\langle s, i, o, s'\rangle \in \lambda_\mathcal{S}$ and $\langle x, i, o, x'\rangle \in \lambda_\mathcal{X}$
                    Add $\langle r, i, o, r'\rangle$ to $\lambda_\mathcal{R}$
                    Define output delays:
                    if $\sigma_\mathcal{S}(\langle s, i, o, s'\rangle) \cap \sigma_\mathcal{X}(\langle x, i, o, x'\rangle) \neq \emptyset$ **then**
                        $\sigma_\mathcal{R}(\langle r, i, o, r'\rangle) = \sigma_\mathcal{S}(\langle s, i, o, s'\rangle) \cap \sigma_\mathcal{X}(\langle x, i, o, x'\rangle)$
                    else $\sigma_\mathcal{R}(\langle r, i, o, r'\rangle) = \sigma_\mathcal{X}(\langle x, i, o, x'\rangle)$
                    end if
                end for
            else
                for each $o \in outputs_X(x,i)$ **do**
                    $r' = \langle\langle s', 0\rangle, \langle x', 0\rangle\rangle$, where $\langle x, i, o, x'\rangle \in \lambda_\mathcal{X}$ and
                    either $\langle s, i, o', s'\rangle \in \lambda_\mathcal{S}$ for some $o \neq o' \notin outputs_X(x,i)$ or $s' = s$.
                    Add $\langle r, i, o, r'\rangle$ to $\lambda_\mathcal{R}$
                    Define output delays: $\sigma_\mathcal{R}(\langle r, i, o, r'\rangle) = \sigma_\mathcal{X}(\langle x, i, o, x'\rangle)$
                end for
            end if
        else
            for each $i \in inputs_X(x)$ **do**
                for each $o \in outputs_X(x,i)$ **do**
                    $r' = \langle\langle s, 0\rangle, \langle x', 0\rangle\rangle$, where $\langle x, i, o, x'\rangle \in \lambda_\mathcal{X}$
                    Add $\langle r, i, o, r'\rangle$ to $\lambda_\mathcal{R}$
                    Define output delays: $\sigma_\mathcal{R}(\langle r, i, o, r'\rangle) = \sigma_\mathcal{X}(\langle x, i, o, x'\rangle)$
                end for
            end for
        end if
    end for
    Define timeout for r:
    $T = \min(\Delta_\mathcal{S}(s)_{\downarrow\mathbb{N}} - t_S, \Delta_\mathcal{X}(x)_{\downarrow\mathbb{N}} - t_X)$
    $r' = \langle\langle time_S(s, t_S{+}T), clock_S(s, t_S{+}T)\rangle, \langle time_X(x, t_X{+}T), clock_X(x, t_X{+}T)\rangle\rangle$
    Define $\Delta_\mathcal{R}(r) = (r', T)$
    if $r' \notin R$ **then** add r' to R and mark not explored.
    end if
    Mark r as explored and take next unexplored $r \in R$.
end for

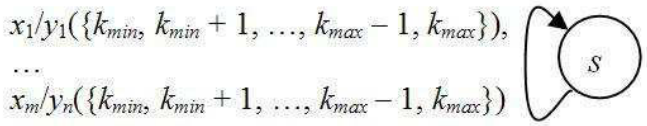

FIGURE 6.1: The maximal TFSM over alphabets $I_X = \{x_1 \ldots x_m\}$ and $O_X = \{y_1 \ldots y_n\}$ with the output delays no less than k_{min} and no greater than k_{max}

Given the largest solution $\mathcal{S}_X$ with alphabets I_X and O_X, the largest solution with output delays not greater than k is the TFSM $\mathcal{S}_{X,\leq k}$ with the language $L(\mathcal{S}_{X,\leq k}) = L(\mathcal{S}_X) \cap (\mathtt{1}^* I_X \mathtt{1}^{\leq k} O_X)^* \mathtt{1}^*$.

In other words, given the largest solution $\mathcal{S}_X$ and the maximal TFSM $\mathcal{M}_{I_X,O_X}$ with maxmin restrictions on output delays values, the largest restricted solution is the intersection $\mathcal{S}_X \cap \mathcal{M}_{I_X,O_X}$. An example of such maximal TFSM is shown in Fig. 6.1.

2. Second, the solutions with upper and lower restrictions on output delays values chosen for each transition individually.

In the latter case, intersection with the maximal TFSM is not enough, and transitions of the largest solution should be explored explicitly.

Since the problem of extracting reduction with required properties is known to be quite complex, similar to classical FSMs, for minimal and maximal output delays on transitions we extract sub-machines rather then reductions.

Choosing among existing option the ones with minimal output delays is aimed on minimizing output delays in composition.

Though, in practice it might be more interesting to choose the component with maximal output delays, since our solution - it is not a component implementation which parameters we actually can enforce, but rather a recommended specification and requirements for the component to be searched for in order to provide the desired composition. Moreover, the dynamic nature of the real networks environment should also be taken into account. The maximal output delays provide the most relaxed and least restrictive requirements for the components - and usage of this result for component implementation/discovery may enable bigger choice of actual service components that might be used in the composition.

6.5 Manipulating component number of states

The TFSM, representing the largest solution of the equation, has two types of transitions: transitions, that are involved in communication with context and satisfying the specification of composition, and transitions, that are not reachable from context and therefore can be defined in any way suitable. The latter might be used for manipulating the number of states in the extracted components, as well as for manipulating the values of timeouts in some states of the component, which allows to make it either more or less tolerant towards timed properties of the context component.

6.5.1 TFSM state minimization

Definition 6.3. The TFSM $\mathcal{S}$ is called *reduced*, if all states are pair-wise distinguishable.

The TFSM $\mathcal{S}$ is called *minimal* if it is reduced and there is no other equivalent TFSM with fewer states.

Given TFSM $\mathcal{P}$, the minimal TFSM $\mathcal{S}$ that is equivalent to $\mathcal{P}$ is called the *minimal form* of the TFSM $\mathcal{P}$. △

For deterministic classical FSMs it is known [70], that the minimal form is unique (up to isomorphism), while for TFSMs it is not necessarily the case.

One of the ways to perform TFSM state minimization is based on deriving the corresponding minimal classical FSM [83], though, the further timeout manipulations issues are not considered on the approach.

Definition 6.4. Given states s and p, state p is called a *copy* of s if $inputs_S(s) = inputs_S(p)$, and $\langle p, i, o, s' \rangle \in \lambda_{\mathcal{S}}$ if and only if $\langle s, i, o, s' \rangle \in \lambda_{\mathcal{S}}$ and $\sigma_{\mathcal{S}}(\langle p, i, o, s' \rangle) = \sigma_{\mathcal{S}}(\langle s, i, o, s' \rangle)$. I.e., if the state p has the same transitions as the state s except, maybe, for timeout transition. △

Another way of deriving some minimal form of TFSM is based on following three main steps:

1. Derive the partition of the set of states induced by the equivalence relation.

Detailed discussion of how to derive the partitioning induced by the equivalence falls out of the scope of this thesis, assuming some adaptation of classical algorithms. For example, for iterative split of equivalence classes according to k-equivalence relation, such adaptation is made by adding a condition on timeout function: at k's step, two states s_1 and s_2 stay in the same equivalence class if $time_S(s_1, k)$ and $time_S(s_2, k)$ were in the same 1-equivalence class (i.e., were in the same class at initial step, distinguishing states by the outputs).

2. Merge all the timeout transitions traversing the chains of "copies"(Definition 6.4).

   In other words, for all $s_1, s_2 \in S$ such that $\Delta_S(s_1) = (s_2, T_1)$, $\Delta_S(s_2) = (s_3, T_2)$ and s_2 is a copy of s_1, redefine $\Delta_S(s_1) = (s_3, T_1 + T_2)$. And if s_2 was reachable only by timeout from s_1, then s_2 becomes unreachable from the initial state and, therefore, should be deleted:

3. Delete all states unreachable from the initial state.

Therefore, based on the step 2, timeouts can be used as a key for state number manipulations in TFSMs while preserving the functional and timed requirements.

Remark 6.5. The considered algorithm is oriented to complete machines, though, might be adapted to incompletely specified ones similar to classical FSMs (see, for example, [84, 85]). △

6.5.2 Manipulating timeout values in equation solutions

The transitions in the largest solution TFSM can be divided in two types:

1. Transitions, that define desired functionality and are required/allowed by communication with the context;

2. Transitions, that are not reachable from the context and, hence, don't care transitions.

And while the former might be necessary to provide the desired behavior of composition TFSM, the latter can be manipulated in any desired way without violating the composition specification.

For instance, correlating don't care transitions with the specified ones, we might extract component with greater or lesser values of timeouts.

The states, for which it is possible to manipulate timeout values, should satisfy the following conditions.

- Consider the state s and with transitions defined on inputs $inputs_S(s)$, the timeout in the state s is finite: $\Delta_{\mathcal{S}}(s) = (s_1, T)$, $T < \infty$.
- If for all $i \in inputs_S(s)$ the state s_1 has only don't care transitions, then the transitions from the state s_1 are redefined so that s_1 becomes a copy of s.
- Then, the timeout for the state s is redefined as $\Delta_{\mathcal{S}}(s) = (s_2, T + T_1)$, where $\Delta_{\mathcal{S}}(s_1) = (s_2, T_1)$, and $\Delta_{\mathcal{S}}(s) = (s, \infty)$ if $T_1 = \infty$.

As a result, the component with close-to-minimal number of states is extracted from the largest solution, and such component might be considered as more tolerant for the context quality violations, preserving the same functional behavior within longer intervals of time.

In case of a less tolerant component is desired, one way to achieve it is to define all transitions that are not reachable from the context to be transitions with error messages, resetting the component due to time requirement violations and, possibly, asking user to repeat the request. For example, if booking of a flight via the Internet is not paid before the expiration of timeout, the booking is canceled, the reserved places are made available for other bookings, and the payment is withdrawn.

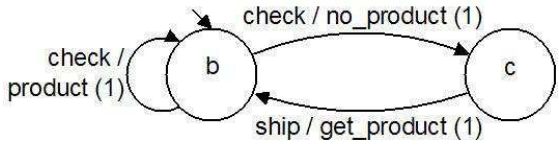

FIGURE 6.2: The restricted solution for the warehouse service with maximized timeouts

Example 6.2. *Let us consider the largest solution for the warehouse service in Fig. 5.3.*

State "a" has don't care transitions and reaches state "b"after the timeout, therefore, we redefine transitions in state "a"so that state "b"becomes a copy of state "a". Then, we do the same with the state "f", which reaches state "a"by timeout. Similar, we combine states "c"and "d". Then, choosing output delays to be minimal non-zero delays (1 in this case), we obtained the restricted solution in Fig. 6.2 that has only infinite timeouts.

However, since we used the largest solution that is not livelock-safe, the further check of livelock-safety might be required. Though, if the solution would've been extracted from the livelock-safe largest solution, it would also be livelock-safe and would require only the check on equivalence of the composition and specification. △

Algorithm 6.2 Extracting the solution with maximized timeouts

Input: The largest solution $\mathcal{S}_X = \langle X, I_X, O_X, x_0, \lambda_\mathcal{X}, \Delta_\mathcal{X}, \sigma_\mathcal{X} \rangle$
Output: The restrcited solution $\mathcal{R} = \langle R, I_X, O_X, r_0, \lambda_\mathcal{R}, \Delta_\mathcal{R}, \sigma_\mathcal{R} \rangle \leq \mathcal{S}_X$.

Assign $\mathcal{R} = \mathcal{S}_X$.
for each state $r \in R$ **do**
  if $\Delta_\mathcal{R}(r) = (r', T)$ where $T < \infty$ **then**
    if $inputs_R(r') \subseteq inputs_R(r)$
      and for all $i \in inputs_R(r)$ it holds $outputs_R(r, i) = O$
    then
      Delete all the transitions starting from r
      For each $\langle r', i, o, r'' \rangle \in \lambda_\mathcal{R}$,
      add $\langle r, i, o, r'' \rangle$ to $\lambda_\mathcal{R}$, with $\sigma_\mathcal{R}(\langle r, i, o, r'' \rangle) = \sigma_\mathcal{R}(\langle r', i, o, r'' \rangle)$
      Re-define $\Delta_\mathcal{R}(r) = (r_T, T_{sum})$ where $\Delta_\mathcal{R}(r') = (r_T, T')$ and $T_{sum} = T + T'$
      if r' is not input-reachable **then**
        Delete r' together with it's all outgoing transitions.
      end if
    end if
  end if
end for

6.6 Chapter conclusions

For some restrictions required from the solution, like the upper- or lower bound of the output delays, the largest solution satisfying the restriction may be derived as an intersection of the general largest solution with the maximal TFSM representing the restriction. Such restrictions include maxmin requirements for the output delays and intervals between applying the next input, forbidden input sequences (in this case maximal TFSM is partial), restriction on the output responses options for certain inputs and input options after producing certain outputs (though, the latter might violate the assumption of harmonized traces, and therefore should be considered with great care).

Other restrictions, like choosing the minimal output delay among allowed options for each transition, cannot be expressed with a maximal TFSM, and might require explicit traversing of transitions in question.

In the chapter we discussed variants of restricted solutions of both types that might be useful in the service composition optimization. Though, the efficient application of provided solutions for discovery and binding of the real services is yet a question to be thoroughly investigated in the future work.

Chapter 7

Conclusions

7.1 Summary of Contributions

Automata theory and its extensions offer powerful and efficient instrumentation for solving various software and digital devices related problems, including the problems from the emerging domain of "everything-as-a-service".

The contributions of the performed work are mainly theoretical. However, preliminary performance experiments with the prototype tool, implementing the construction of TFSM parallel composition, show that machines with hundreds states might be processed efficiently despite the issue of state explosion while transforming from TFSMs to corresponding finite automata.

The correlation of service composition optimization issues to the optimization of finite state model composition allows to integrate quality- and safety-aware design and testing of composite services based on formal specifications.

To sum up, the main contributions include:

1. The nondeterministic extension of FSM with Timeouts (TFSM) model for service description integrating time-related quality and safety parameters.

2. A correspondence between functional conformance relations for service modeled by TFSMs in cases of real and integer-valued time variable has been established.

3. A method for iterative optimization of service composition with dialog-mode communication between components, based on deriving the largest solution for the corresponding TFSM parallel equation.

4. A method for minimizing the set of composite service requirements with respect to the component under optimization, which can be used to shorten the testing and verification of the composition after the substitution of the component.

5. Techniques for extracting restricted solutions with required properties (livelock/deadlock-safety, number of states and timeout values, least or most restrictive output delays requirements, similarity to the initial behavior of the component under optimization).

7.2 Perspectives and Future Work

One of the serious drawbacks of the approach we are proposing, lays within its scalability towards multiple interacting components. The parallel composition operator we rely on focuses on the dialog-mode interactions and tracks the interactions when at each moment of time only one component is considered working. For orchestrations, such concept might be enough as far as parallel execution is not in need; then, the orchestrator communication on request-response basis with the pool of component services, leading them through the desired workflow, is represented as a parallel composition of the orchestrator and the pool. For scenario-based user-driven choreographies the parallel composition also might be applied, assuming that the component in work is the component currently in charge of user-service interaction.

However, the current definition of the operator does not capture independent parallel execution with further concurrency and/or synchronization of the processes, and for the future work should be therefore updated.

Discussion on capabilities of the model itself to integrate the quality parameters other than time is also an interesting topic for further research. For cumulative quality parameters, e.g. cost, all the results obtained in the thesis, might be applied with some minor changes. Timeouts, in such case, are interpreted as the boundary conditions for different levels of quality, changing the responses of the service depending on the

quality value. In composition, the correlation between components on quality parameters is made between output delays and timeouts, and for additive parameters this correlation is preserved: output delays represent the consumption of quality parameter for operation, and timeouts in another components ensure responses depending on this consumption. Therefore, the correspondence of TFSMs to finite automata can be used in composition with TFSM timer interpreted as any other additive parameter, without major adjustments.

Though, direct adaptation of the timeouts to other types of quality parameters is not obvious and require further research. It seems, that methods over the model and composition may be preserved for other types of parameters in place of time, using timeouts as conditions and output delays as applied values of parameter, but with different definition for aggregation of parameter values in composition. Cost, like time, is an additive parameter: having sequences of internal operations, the resulting value is obtained as a sum of partial values. That is the basis that allows to unroll each transition under k time units into a chain of k transitions for a single `1` unit, as well as then fold it back together without loosing value in translation. For other types of parameters, multiplicative or minmax, for instance, such transformation requires thorough investigation of a unit value and unrolling concept, that would allow to use finite automata operations without additional symbolic refinements.

To summarize, the main directions of the future work include the extension of the composition operator for more complex modes of communication, including multi-stimuli composition, and investigation of properties of the model to integrate other than additive quality parameters within the same framework.

Bibliography

[1] Eyhab Al-Masri and Qusay H. Mahmoud. Investigating web services on the world wide web. *Proceedings of the 17th international conference on World Wide Web*, pages 795–804, 2008. doi: 10.1145/1367497.1367605. 1367605 795-804.

[2] Ade McCormack. Cloud – everything as a service? Report, 2013. URL http://www8.hp.com/uk/en/pdf/Auridian_Paper2_aw_High_tcm_183_1326448.pdf.

[3] Google app engine: Platform as a service. URL https://cloud.google.com/appengine/docs.

[4] Amazon web services. URL http://aws.amazon.com/.

[5] Mathew Sajee. Overview of amazon web services, 2014. URL http://d0.awsstatic.com/whitepapers/aws-overview.pdf.

[6] Steve Bratt. Web 3.0 emerging, 2007. URL http://www.w3.org/2007/Talks/0123-sb-W3CEmergingTech/#(2).

[7] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, Nirmal Mukhi, and S. Weerawarana. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *Internet Computing, IEEE*, 6(2):86–93, 2002. ISSN 1089-7801. doi: 10.1109/4236.991449.

[8] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web services architecture, 2004. URL http://www.w3.org/TR/ws-arch/.

[9] The Open Group SOA Working Group. Service oriented architecture: What is soa? URL http://www.opengroup.org/soa/source-book/soa/soa.htm.

[10] Stas Khirman and Peter Henriksen. Relationship between quality-of-service and quality-of-experience for public internet service. *Proceedings of the Passive and Active Network Measurement Workshop PAM02*, 2002. doi: citeulike-article-id:8507159. URL http://www.pamconf.net/2002/Relationship_Between_QoS_and_QoE.pdf.

[11] Kim Hyun-Jong, Lee Dong Hyeon, Lee Jong Min, Lee Kyoung-Hee, Lyu Won, and Choi Seong-Gon. The qoe evaluation method through the qos-qoe correlation model. *Proceedings of the 4th International Conference on Networked Computing and Advanced Information Management, NCM 08*, 2:719–725, 2008. doi: 10.1109/NCM.2008.202.

[12] Anderson Morais and Ana Cavalli. Deliverable d2.1 – state of the art of sqm/cem technology, tools, and standartization. Report, 2012. URL http://projects.celtic-initiative.org/ipnqsis/IPNQSIS-D21.pdf.

[13] C.K. Fung, P.C.K. Hung, R.C. Linger, and G.H. Walton. Extending business process execution language for web services with service level agreements expressed in computational quality attributes. In *System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on*, pages 166a–166a, Jan 2005. doi: 10.1109/HICSS.2005.268.

[14] F. Rosenberg, C. Enzi, A. Michlmayr, C. Platzer, and S. Dustdar. Integrating quality of service aspects in top-down business process development using ws-cdl and ws-bpel. In *Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International*, pages 15–15, Oct 2007. doi: 10.1109/EDOC.2007.23.

[15] H. Kreger. Web services conceptual architecture (wsca 1.0). Technical report, IBM Software Group, May 2001. URL http://www.csd.uoc.gr/~hy565/newpage/docs/pdfs/papers/wsca.pdf.

[16] Pablo Rabanal, Ismael Rodriguez, Jose A. Mateob, and Gregorio Diaz. Improving the automatic derivation of choreography-conforming web services systems. In *Proceedings of the International Conference on Computational Science, ICCS 2012*, volume 9, page 449–458, 2012. ISBN - 1877-0509. doi: -http://dx.doi.org/10.1016/j.procs.2012.04.048.

[17] Samik Basu, Tevfik Bultan, and Meriem Ouederni. Deciding choreography realizability. *SIGPLAN Not.*, 47(1):191–202, 2012. ISSN 0362-1340. doi: 10.1145/2103621.2103680.

[18] Ismael Rodriguez, Gregorio Diaz, Pablo Rabanal, and Jose Antonio Mateo. A centralized and a decentralized method to automatically derive choreography-conforming web service systems. *The Journal of Logic and Algebraic Programming*, 81(2):127 – 159, 2012. ISSN 1567-8326. doi: http://dx.doi.org/10.1016/j.jlap.2011.10.001. URL http://www.sciencedirect.com/science/article/pii/S1567832611000762. Formal Languages and Analysis of Contract-Oriented Software (FLACOS'10).

[19] Pablo Rabanal, Ismael Rodriguez, Jose A. Mateo, and Gregorio Diaz. Improving the automatic derivation of choreography-conforming web services systems. *Procedia Computer Science*, 9(0):449–458, 2012. ISSN 1877-0509. doi: http://dx.doi.org/10.1016/j.procs.2012.04.048. URL http://www.sciencedirect.com/science/article/pii/S187705091200169X.

[20] Gregorio Diaz, Juan-José Pardo, María-Emilia Cambronero, Valentín Valero, and Fernando Cuartero. *Automatic Translation of WS-CDL Choreographies to Timed Automata*, volume 3670 of *Lecture Notes in Computer Science*, book section 17, pages 230–242. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-28701-8. doi: 10.1007/11549970_17. URL http://dx.doi.org/10.1007/11549970_17.

[21] Pierre-Cyrille Héam, Olga Kouchnarenko, and Jérôme Voinot. Component simulation-based substitutivity managing qos and composition issues. *Science of Computer Programming*, 75(10):898–917, 2010. ISSN 0167-6423. doi: http://dx.doi.org/10.1016/j.scico.2010.02.004. URL http://www.sciencedirect.com/science/article/pii/S0167642310000316.

[22] Ching-Seh Wu and Chi-Hsin Huang. The web services composition testing based on extended finite state machine and uml model. In *Service Science and Innovation (ICSSI), 2013 Fifth International Conference on*, pages 215–222, May 2013. doi: 10.1109/ICSSI.2013.46.

[23] M. Lallali, F. Zaidi, and A. Cavalli. Timed modeling of web services composition for automatic testing. In *Signal-Image Technologies and Internet-Based System, 2007.*

SITIS '07. Third International IEEE Conference on, pages 417–426, Dec 2007. doi: 10.1109/SITIS.2007.110.

[24] A. Sahai and S. Graupner. *Web Services in the Enterprise: Concepts, Standards, Solutions, and Management*. Kluwer Academic/Plenum Publishers network and systems management. Springer, 2007. ISBN 9780387275970. URL http://books.google.fr/books?id=zU0CzkY_0FoC.

[25] Service orientation. URL http://serviceorientation.com/.

[26] Web service depository xmethods. URL http://www.xmethods.net.

[27] P. Reichl, S. Egger, R. Schatz, and A. DAlconzo. The logarithmic nature of qoe and the role of the weber-fechner law in qoe assessment. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1–5, 2010. ISBN 1550-3607. doi: 10.1109/ICC.2010.5501894.

[28] Sherry X. Sun and Jing Zhao. A decomposition-based approach for service composition with global qos guarantees. *Information Sciences*, 199(0):138–153, 2012. ISSN 0020-0255. doi: http://dx.doi.org/10.1016/j.ins.2012.02.061. URL http://www.sciencedirect.com/science/article/pii/S0020025512001892.

[29] Jorge Cardoso, Amit Sheth, John Miller, Jonathan Arnold, and Krys Kochut. Quality of service for workflows and web service processes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(3):281–308, 2004. ISSN 1570-8268.

[30] J. El Hadad, M. Manouvrier, and M. Rukoz. Tqos: Transactional and qos-aware selection algorithm for automatic web service composition. *Services Computing, IEEE Transactions on*, 3(1):73–85, 2010. ISSN 1939-1374. doi: 10.1109/TSC.2010.5.

[31] C. C. Chang and Lu Hsueh-Ming. Integration of heterogeneous medical decision support systems based on web services. In *Bioinformatics and BioEngineering, 2009. BIBE 09. Ninth IEEE International Conference on*, pages 415–422, 2009. doi: 10.1109/BIBE.2009.59.

[32] B. Pernici and S. H. Siadat. Evaluating web service qos: A neural fuzzy approach. *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA2011)*, pages 1–6, 2011. doi: 10.1109/SOCA.2011.6166267.

[33] LeonardoA.F. Leite, Gustavo Ansaldi Oliva, GuilhermeM. Nogueira, MarcoAurélio Gerosa, Fabio Kon, and DejanS. Milojicic. A systematic literature review of service choreography adaptation. *Service Oriented Computing and Applications*, 7(3):199–216, 2013. ISSN 1863-2386. doi: 10.1007/s11761-012-0125-z. URL http://dx.doi.org/10.1007/s11761-012-0125-z.

[34] Alex Norta, Lixin Ma, Yucong Duan, Addi Rull, Merit Kõlvart, and Kuldar Taveter. econtractual choreography-language properties towards cross-organizational business collaboration. *Journal of Internet Services and Applications*, 6(1), 2015. ISSN 1867-4828. doi: 10.1186/s13174-015-0023-7. URL http://dx.doi.org/10.1186/s13174-015-0023-7.

[35] Alistair Barros. *Process Choreography Modelling*, pages 279–300. International Handbooks on Information Systems. Springer Berlin Heidelberg, 2015. ISBN 978-3-642-45099-0. doi: 10.1007/978-3-642-45100-3_12. URL http://dx.doi.org/10.1007/978-3-642-45100-3_12.

[36] Carlos Rodríguez-Domínguez, Tomás Ruiz-López, JoséLuis Garrido, Manuel Noguera, and Kawtar Benghazi. *Leveraging the Model-Driven Architecture for Service Choreography in Ubiquitous Systems*, volume 8276 of *Lecture Notes in Computer Science*, pages 303–310. Springer International Publishing, 2013. ISBN 978-3-319-03175-0. doi: 10.1007/978-3-319-03176-7_39. URL http://dx.doi.org/10.1007/978-3-319-03176-7_39.

[37] Kashif Dar, Amirhosein Taherkordi, Roman Vitenberg, Romain Rouvoy, and Frank Eliassen. Adaptable service composition for very-large-scale internet of things systems. In *Proceedings of the Workshop on Posters and Demos Track*, PDT '11, pages 11:1–11:2, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1073-4. doi: 10.1145/2088960.2088971. URL http://doi.acm.org/10.1145/2088960.2088971.

[38] Valérie Issarny, Antonia Bertolino, Guglielmo De Angelis, Amira Ben Amida, Jean-Pierre Lorré, Nikolaos Georgantas, Animesh Pathak, James Lockerbie, and et al. Deliverable d1.1: Choreos state of the art, baseline, and beyond. Report, 2011. URL http://www.choreos.eu/bin/download/Share/Deliverables/CHOReOS-StateoftheArt-BaselineandBeyond-VA.0.pdf.

[39] Lina Bentakouk, Pascal Poizat, and Fatiha Zaïdi. Checking the behavioral conformance of web services with symbolic testing and an smt solver, 2011. 2025940 33-50.

[40] ISO/IEC. Software life cycle processes, 1995.

[41] Rong Huigui, zhou Ning, Chen HongQin, and Cheng Hongli. Research on strategy of web service composition based on software life cycle. *Proceedings of 4th International Conference on Wireless Communications, Networking and Mobile Computing, WiCOM 08*, pages 1–4, 2008. doi: 10.1109/WiCom.2008.2002.

[42] Diego R. Ferreira. Business networking with web services: Supporting the full life cycle of business collaborations. *Electronic Commerce: Concepts, Methodologies, Tools, and Applications*, pages 2225–2239, 2008. ISSN 9781599049434. doi: 10.4018/978-1-59904-943-4.ch170. URL http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-59904-943-4.ch170.

[43] Shu Zhang and Meina Song. An architecture design of life cycle based sla management. *Proceedings of the 12th International Conference on Advanced Communication Technology (ICACT)*, 2:1351–1355, 2010. ISSN 1738-9445.

[44] Jose Pablo Escobedo Del Cid. *Symbolic test case generation for testing orchestrators in context.* Thesis, 2011.

[45] Christophe Gaston and Pascal Le Gall. About incremental model-based testing of web service orchestrations, 2012. URL http://tarot2012.univ-fcomte.fr/?talks.

[46] Zhao Xin, Shen Li Wei, Peng Xin, and Zhao Wenyun. Finding preferred skyline solutions for sla-constrained service composition. *IEEE 20th International Conference on Web Services (ICWS13)*, pages 195–202, 2013. doi: 10.1109/ICWS.2013.35.

[47] F. Lalanne, A. Cavalli, and S. Maag. Quality of experience as a selection criterion for web services. *Proceedings of the 8th International Conference on Signal Image Technology and Internet Based Systems (SITIS12)*, pages 519–526, 2012. doi: 10.1109/SITIS.2012.81.

[48] Zhang Shaoqian, Dou Wanchun, and Chen Jinjun. Selecting top-k composite web services using preference-aware dominance relationship. *Proceedings of the IEEE*

20th International Conference on Web Services (ICWS13), pages 75–82, 2013. doi: 10.1109/ICWS.2013.20.

[49] Maurice Beek, Antonio Bucchiarone, and Stefania Gnesi. A survey on service composition approaches: From industrial standards to formal methods, 2006. URL http://fmt.isti.cnr.it/WEBPAPER/TRWS-FM06.pdf.

[50] Cao Tien-Dung, P. Felix, and R. Castanet. Wsotf: An automatic testing tool for web services composition. *Proceedings of the 5th International Conference on Internet and Web Applications and Services*, pages 7–12, 2010. doi: 10.1109/ICIW.2010.9.

[51] Gao Honghao and Li Ying. Generating quantitative test cases for probabilistic timed web service composition. *IEEE Asia-Pacific Services Computing Conference (APSCC)*, pages 275–283, 2011. doi: 10.1109/APSCC.2011.13.

[52] Alexander Keller and Heiko Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. Report, IBM, 2002.

[53] Huiyuan Zheng, Weiliang Zhao, Jian Yang, and Athman Bouguettaya. Qos analysis for web service compositions with complex structures. *Services Computing, IEEE Transactions on*, 6(3):373–386, 2013. ISSN 1939-1374. doi: 10.1109/TSC.2012.7.

[54] E. Billionniere, D. Greiman, and K. Gosha. A comparison of social service selection techniques. *Proceedings of the Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing, DASC09*, pages 260–265, 2009. doi: 10.1109/DASC.2009.24.

[55] Deng Xiaopeng and Xing Chunxiao. A qos-oriented optimization model for web service group. *Proceedings of the 8th IEEE/ACIS International Conference on Computer and Information Science, ICIS09*, pages 903–909, 2009. doi: 10.1109/ICIS.2009.91.

[56] M. C. Jaeger, G. Rojec-Goldmann, and G. Muhl. Qos aggregation for web service composition using workflow patterns. *Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2004*, pages 149–159, 2004. ISSN 1541-7719. doi: 10.1109/EDOC.2004.1342512.

[57] Yu Tao and Lin Kwei-Jay. Service selection algorithms for web services with end-to-end qos constraints. *Proceedings of the IEEE International Conference on e-Commerce Technology, CEC 2004*, pages 129–136, 2004. doi: 10.1109/ICECT.2004.1319726.

[58] Du Wu and Fan Hong. An automatic service composition algorithm for constructing the global optimal service tree based on qos. *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 3976–3979, 2010. ISSN 2153-6996. doi: 10.1109/IGARSS.2010.5650959.

[59] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng. Quality driven web services composition. *Proceedings of the 12th international conference on World Wide Web*, pages 411–421, 2003. doi: 10.1145/775152.775211. 775211 411-421.

[60] O. Moser, F. Rosenberg, and S. Dustdar. Domain-specific service selection for composite services. *Software Engineering, IEEE Transactions on*, 38(4):828–843, 2012. ISSN 0098-5589. doi: 10.1109/TSE.2011.43.

[61] J. Mtsweni. Exploiting uml and acceleo for developing semantic web services. *Proceedings of the International Conference For Internet Technology And Secured Transactions, 2012*, pages 753–758, 2012.

[62] M. Lallali, F. Zaidi, A. Cavalli, and Hwang Iksoon. Automatic timed test case generation for web services composition. *Proceedings of the IEEE 6th European Conference on Web Services, ECOWS 08*, pages 53–62, 2008. doi: 10.1109/ECOWS.2008.14.

[63] Ethical IT. Example sla v1, 2008. URL http://www.ictknowledgebase.org.uk/fileadmin/ICT/pdf/support_contracts/Ethical_IT_Example_SLA_v1.pdf.

[64] M. G. Merayo, M. Nunez, and I. Rodriguez. Extending efsms to specify and test timed systems with action durations and time-outs. *Computers, IEEE Transactions on*, 57(6):835–844, 2008. ISSN 0018-9340. doi: 10.1109/TC.2008.15.

[65] Maxim Gromov, Khaled El-Fakih, Natalia Shabaldina, and Nina Yevtushenko. *Distinguishing Non-deterministic Timed Finite State Machines*, volume 5522 of

Lecture Notes in Computer Science, pages 137–151. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-02137-4. doi: 10.1007/978-3-642-02138-1_9. URL http://dx.doi.org/10.1007/978-3-642-02138-1_9.

[66] M. Zhigulin, N. Yevtushenko, S. Maag, and A. Cavalli. Fsm-based test derivation strategies for systems with time-outs. In *Quality Software (QSIC), 2011 11th International Conference on*, pages 141–149, 2011. ISBN 1550-6002. doi: 10.1109/QSIC.2011.30.

[67] Davide Bresolin, Khaled El-Fakih, Tiziano Villa, and Nina Yevtushenko. Deterministic timed finite state machines: Equivalence checking and expressive power. In Adriano Peron and Carla Piazza, editors, *Proceedings Fifth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2014, Verona, Italy, September 10-12, 2014.*, volume 161 of *EPTCS*, pages 203–216, 2014. doi: 10.4204/EPTCS.161.18. URL http://dx.doi.org/10.4204/EPTCS.161.18.

[68] Alexandre Petrenko and Nina Yevtushenko. Conformance tests as checking experiments for partial nondeterministic fsm. In Wolfgang Grieskamp and Carsten Weise, editors, *Formal Approaches to Software Testing*, volume 3997 of *Lecture Notes in Computer Science*, pages 118–133. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-34454-4. doi: 10.1007/11759744_9. URL http://dx.doi.org/10.1007/11759744_9.

[69] AlexD.B. Alberto and Adenilso Simao. Iterative minimization of partial finite state machines. *Central European Journal of Computer Science*, 3(2):91–103, 2013. ISSN 1896-1533. doi: 10.2478/s13537-013-0106-0. URL http://dx.doi.org/10.2478/s13537-013-0106-0.

[70] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 2nd edition, 2001. ISBN 020102988X. 574901.

[71] Tiziano Villa, Nina Yevtushenko, RobertK Brayton, Alan Mishchenko, Alexandre Petrenko, and Alberto Sangiovanni-Vincentelli. *The Unknown Component Problem*. Springer US, 2012. ISBN 978-0-387-34532-1. doi: 10.1007/978-0-387-68759-9_2. URL http://dx.doi.org/10.1007/978-0-387-68759-9_2.

[72] Javier Esparza, Pierre Ganty, Stefan Kiefer, and Michael Luttenberger. Parikh's theorem: A simple and direct automaton construction. *Inf. Process. Lett.*, 111(12): 614–619, 2011. ISSN 0020-0190. doi: 10.1016/j.ipl.2011.03.019.

[73] Davide Bresolin, Khaled El-Fakih, Tiziano Villa, and Nina Yevtushenko. Deterministic timed finite state machines: Equivalence checking and expressive power. *Presentation at Fifth International Symposium on Games, Automata, Logics and Formal Verification (GandALF), in Verona, Italy, 10th - 12th September 2014*, 2014.

[74] IBM WebSphere Integration Developer. The loan application, version 6.0.2, 2006.

[75] Boston Redevelopment Authority. Application for loan and grant, part ii; local project approval data, 2010.

[76] Olga Kondratyeva, Nina Yevtushenko, and Ana Cavalli. Parallel composition of nondeterministic finite state machines with timeouts. *Journal of Control and Computer Science, Tomsk State University, Russia*, 2(27):73–81, 2014. ISSN ISSN 2311-2085 (Online), ISSN 1998-8605 (Print).

[77] I.B. Bourdonov and A.S. Kossatchev. Interaction semantics with refusals, divergence, and destruction. *Programming and Computer Software*, 36(5): 247–263, 2010. ISSN 0361-7688. doi: 10.1134/S0361768810050014. URL http://dx.doi.org/10.1134/S0361768810050014.

[78] R. Alur. Timed automata. *Verification of Digital and Hybrid System*, 170:233–264, 2000. ISSN 0258-1248. URL <GotoISI>://WOS:000087176700012.

[79] Olga Kondratyeva, Nina Yevtushenko, and Ana Cavalli. Solving parallel equations for finite state machines with timeouts. *Proceedings of the Institute for System Programming*, 26(6):85–98, 2014. ISSN ISSN 2220-6426 (Online), ISSN 2079-8156 (Print). doi: 10.15514/ISPRAS-2014-26(6)-8.

[80] Ricardo Anido, Ana R. Cavalli, Luiz Paula Lima, and Nina Yevtushenko. Test suite minimization for testing in context. *Software Testing, Verification and Reliability*, 13:141–155, 2003. doi: 10.1002/stvr.275.

[81] H.N. Nguyen, P. Poizat, and F. Zaidi. Automatic skeleton generation for data-aware service choreographies. In *Software Reliability Engineering (ISSRE), 2013*

IEEE 24th International Symposium on, pages 320–329, Nov 2013. doi: 10.1109/ISSRE.2013.6698885.

[82] Sébastien Salva and Tien-Dung Cao. A model-based testing approach combining passive conformance testing and runtime verification: Application to web service compositions deployed in clouds. In Roger Lee, editor, *Software Engineering Research, Management and Applications*, volume 496 of *Studies in Computational Intelligence*, pages 99–116. Springer International Publishing, 2014. ISBN 978-3-319-00947-6. doi: 10.1007/978-3-319-00948-3_7. URL http://dx.doi.org/10.1007/978-3-319-00948-3_7.

[83] Alexandre Tvardovskii. On the minimization of timed finite state machines. *Proceedings of the Institute for System Programming*, 26(6):77–84, 2014.

[84] June kyung Rho, Gary D. Hachtel, Fabio Somenzi, and Reily M. Jacoby. Exact and heuristic algorithms for the minimization of incompletely specified state machines. *IEEE Transactions on Computer-Aided Design*, 1994.

[85] A.S. Klimowicz and V.V. Solov'ev. Minimization of incompletely specified mealy finite-state machines by merging two internal states. *Journal of Computer and Systems Sciences International*, 52(3):400–409, 2013. ISSN 1064-2307. doi: 10.1134/S106423071303009X. URL http://dx.doi.org/10.1134/S106423071303009X.

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Stratégie basée sur les machines à états finis temporisées pour optimiser la composition de services web à l'égard de la qualité et de la sécurité

Mots clés : machine temporisée à états finis, composition de services web, optimisation, qualité de service, sécurité

Résumé : Les concepts d'architecture orientée services (SOA) ainsi que tout une panoplie de technologies «en tant que service» (XaaS) sont utilisées quasiment partout de nos jours, et l'organisation optimisée d'activités synchronisées devient un défi important. Dans le but de proposer à l'utilisateur final un service sécuritaire et fiable sans compromettre la qualité, les questions concernant la vérification et la validation des compositions des services deviennent d'un grand intérêt tant théorique que pratique. Dans les autres travaux traitant du sujet, de nombreux modèles et techniques sont proposés, mais la plupart mettent l'accent sur les aspects fonctionnels ou non-fonctionnels pris séparément, alors que l'intégration de ces paramètres en un modèle formel unifié reste un problème qui doit être résolu – ce qui est devenu par conséquent un des objectifs fondamentaux de cette thèse.
Dans notre travail, nous réfléchissons au problème de l'optimisation des compositions des services web. Tout ceci est étudié dans l'optique de la fonctionnalité des systèmes, de leur qualité et de la sécurité des compositions. Il a été prouvé que les modèles à états finis sont utiles à des fins de tests et de vérification, de même que pour le contrôle qualité à chaque étape du développement du service. C'est pour cette raison que nous proposons d'utiliser le modèle de machine temporisée à états finis (TFSM) pour intégrer une description fonctionnelle du service avec les paramètres de sécurité et de qualité liées au temps.
L'extension du modèle permettra alors d'interpréter adéquatement le non-déterminisme significatif causé par un manque d'observabilité ou/et de contrôle sur les services tiers. Dans le but d'optimiser les compositions des systèmes, nous proposons une méthode pour dériver la solution la plus globale contenant tous les composants autorisés pour la mise en œuvre du service, basée sur la résolution de l'équation parallèle du TFSM. Ensuite, les techniques pour extraire des solutions restreintes avec les propriétés requises (paramètres de temps minimisé/maximisé, interblocages actifs ou passifs, similarité avec le composant d'origine donné, etc.) ont été proposées. Dans le cas où les spécifications d'un service composite consistent en un ensemble d'exigences fonctionnelles, éventuellement renforcées par des exigences de qualité, nous proposons une technique de minimisation de l'ensemble, dans le respect du composant à optimiser. L'application des résultats obtenus à la découverte et à la mise en place de composants plus efficaces, ainsi que l'extension du modèle à des modes de communication plus complexes font partie des sujets possibles pour des études futures.

Title : Timed FSM strategy for optimizing web service compositions w.r.t. the quality and safety issues

Keywords : FSM with timeouts, web service composition optimization, quality evaluation, safety

Abstract: Service-oriented architecture (SOA) together with a family of Everything-as-a-Service (XaaS) concepts nowadays are used almost everywhere, and the proper organization of collaborative activities becomes an important challenge. With the goal of bringing to the end-user safe and reliable service with the guaranteed level of quality, issues of service compositions verification and validation become of high practical and theoretical interest. In the related work, numerous models and techniques are proposed, but mostly focused on functional and non-functional issues in isolation, while integration of these parameters within a unified formal framework still remains a problem to be solved – and therefore became one of the core objectives of this thesis.
In our work, we address the problems of web service composition verification and optimization with respect to functionality, quality and safety properties of the composition. Finite state models are proven to be useful for testing and verification purposes as well as for service quality evaluation at each step of service development. Therefore, we propose to use the model of Finite State Machine with Timeouts (TFSM) for integrating functional description with time-related quality and safety parameters, and suggest the extension of the model in order to adequately inherit significant nondeterminism due to the lack of observability and control over third-party component services. For the purpose of component optimization in the composition, we propose a method for deriving the largest solution containing all allowed component service implementations, based on solving TFSM parallel equation. Further, techniques for extracting restricted solutions with required properties (minimized/maximized time parameters, deadlock- and livelock-safety, similarity to the initially given component, etc.) have been proposed. In cases when the specification of a composite service is provided as a set of functional requirements, possibly, augmented with quality requirements, we propose a technique to minimize this set with respect to the component under optimization. Application of the obtained results for more efficient candidate component services discovery and binding, alongside with extending the framework for more complex distributed modes of communications, are among the topics for the future work.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France