En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *25/09/2014* **par :**

Hikmat FARHAT

Synthèse D'orchestrateur Pour La Composition De Services

JURY

Mr Jean-Paul Bodeveix	Professeur -Université Paul Sabatier	Président du Jury
Mme Olga Kouchnarenko	Chargé de Recherche-Université Franche-Comté	Rapportrice
Mr Jean-François Condotta	Enseignant Chercheur-Université d'Artois	Rapporteur
Mr Philipe Balbiani	Chargé de Recherche-Université Paul Sabatier	Directeur de thèse
Mr Guillaume Feuillade	Maitre de Conference-Université Paul Sabatier	Codirecteur de thèse
Mme Fahima Alili-Cheikh	Maitre de Conference-Université d'Artois	Membre du Jury

École doctorale et spécialité :

MITT : Domaine STIC : Intelligence Artificielle

Unité de Recherche :

Institut de Recherche en Informatique de Toulouse

Directeur(s) de Thèse :

Mr Philippe BALBIANI et *Mr Guillaume FEUILLADE*

Rapporteurs :

Mme Olga KOUCHNARENKO et *Mr Jean-François CONDOTTA*

Composition of Services Behavior via Orchestrator Synthesis

Thesis presented and defended by

Hikmat FARHAT

On the 25th of September 2014

To obtain the degree of

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Delivered by: Université Toulouse III Paul Sabatier(UPS)

Speciality: Computer Science

Advisors

Philippe Balbiani
Chargé de Recherche
Université Paul Sabatier

Guillaume Feuillade
Maitre de Conference
Université Paul Sabatier

Reviewers

Olga Kouchnarenko
Chargé de Recherche
Université Franche-Comté

Jean-François Condotta
Maitre de Conference
Université d'Artois

Members of the Jury

Jean-Paul Bodeveix (President)
Professeur
Université Paul Sabatier

Fahima Alili-Cheikh
Maitre de Conference
Université d'Artois

École doctorale:
Mathématiques Informatique Télécommunications de Toulouse (MITT)
Unité de recherche:
Institut de recherche en Informatique de Toulouse (IRIT)

Acknowledgement

First I would like to express my gratitude to my thesis advisor Philippe Balbiani for taking me as a Doctorant in his group.

A big thank you for Guillaume Feuillade for all the time he spent in guiding me throughout my thesis work. He was always patient, insightful, and enthusiastic. I wouldn't have been able to finish my thesis without his help.

I would like to extend my thanks to the Olga Kouchnarenko and Jean-François Condotta for taking the time to review my thesis. I would like also to thank Fahima Alili-Cheikh and Jean-Paul Bodeveix for giving the honor of serving as jury members.

Finally, I thank my wife Samar for her support and encouragement.

Abstract

The behavior composition problem is an important aspect in many fields, especially in Service Oriented Computing and in Multi-agent systems. The basic objective is to orchestrate the behavior of the different available components, modelled as labelled transition systems (LTS), in order to satisfy a given goal specification, also modelled as an LTS . A major concern has been the large state space of typical situations which made existing approaches very compute intensive. The aim of this thesis is to develop efficient methods to solve the behavior composition problem.

First we study the case when all actions are observable. We develop a characterization of the existence of a solution in term of existence of a relation between the different available components, considered as a single system, on one hand and the goal specification on the other. Using that characterization we develop an on-the-fly algorithm that finds a solution to the problem when one exists. The algorithm is shown to be correct and has polynomial complexity with the respect to the size of the components. We also show that the algorithm is robust with respect to component failure.

Then we propose an abstraction method that reduces drastically the number of states. We show that the non-existence of a solution in the abstracted systems implies the non-existence of solution in the original system. Also the result of the abstraction is used as an input to the above algorithm for use as a heuristic to speed up the search.

Finally, we develop a characterization of the behavior composition problem in the case of partial observation by using the concept of controllability. We show that a solution to the composition problem with partial observation exists if and only if the components are controllable with respect to the goal specification. We also develop an on-the-fly algorithm to compute the controllability of the system. The complexity of the algorithm is EXPTIME in the size of the components.

Résumé

La composition de comportement est un aspet important dans beaucoup de domaines , surtout dans la programmation orientèe service (Service Oriented Computing) et dans les systèmes Multi-agents. L'objectif est d'orchestrer le comportement des différents modules, modelisés as labelled transition systems (LTS), pour satisfaire une specification but, de même modelisé as an LTS. Un problême majeur qu'on trouve dans la plupart des approches, est le nombre élevé d'états. L'objectif de cette thèse est de développer des méthodes efficaces pour résoudre le problème de composition de comportement.

D'abord on analyse le cas où toutes les actions sont observables. Nous developpons ensuite une caractérisation de l'existence d'une solution en terme d'une relation entre les différentes composantes d'une part et la specification but d'autre part. En utilisant cette caractérisation , nous développons un algorithme qui trouve à la volée une solution au problème si cette dernière existe. Nous démontrons que l'algorithme est correct et sa complexité est polynomial part respect. Nous prouvons également que l'algorithme est robuste par rapport à l'échec d'un des composants .

Ensuite, nous proposons une méthode d'abstraction qui réduit considérablement le nombre d'états. Cette abstraction est utilisée comme outil heuristique qui accélère la recherche.

Finalement, nous développons une caractérisation de l'existence d'une solution dans le cas d'observation partielle. Cette caractérisation est elaborée en introduisant le concept de contrabilité. Nous démontrons qu'une solution existe si et seulement si les composantes sont controlables par rapport au but. Nous developpons un algorithme pour trouver la relation de controlabilité à la volée. La complexité de l'algorithme est EXPTIME en terme de la taille des composantes.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

Contents

1.1 Service Oriented Computing

Service Oriented Computing (SOC) [ACKM04] is a programming paradigm that uses individual reusable components, called services, to support rapid development of distributed applications in a heterogeneous environment. Since any application can be wrapped by an additional layer and presented as a service, this paradigm promises a flexible computing architecture independent of any specific technology. To enable this, a service should have an interface that is independent from implementation. In addition to being self-contained, and platform-independent, SOC requires services to be dynamically discoverable, invokable and composable. The latter property is what interests us most in this thesis because, being composable, a collection of services can be combined in such a way as to produce a result, none of the individual services can produce by itself.

Usually, there are two ways to compose web services: by *orchestration* or by *choreography*. Orchestration requires a centralized process, in which an entity called an *orchestrator*, coordinates the execution of different web services. The orchestrator sends and receives message from services without each individual service being necessarily aware of the overall goal. The timing with which a given service is invoked is decided on by the orchestrator. Choreography, on the other hand, does not rely on an orchestrator, but it is a collaborative effort where each service knows exactly what actions to perform and when to perform them. All services are aware of the overall goal and they interact together to achieve that goal. In this thesis were are concerned with *orchestration* and more specifically how to synthesize an *orchestrator* process, such that community of services under the control of the orchestrator, satisfies a given goal. This orchestration synthesis task is referred to as the behavior composition problem.

The behavior composition problem has been subject to intensive research. This fact can be seen from the widely different approaches for the composition problem, ranging from model checking [FVKR11], agent planning [DGS07], satisfiability solving [ZPG12], and theorem proving [PF11] (see [RS05] for a survey). The framework we use in this paper, is similar to the one first proposed in [BCG$^+$03], usually referred to as the "Roman Model", and has been dealt with in many related works [BCF08a][DGP10][BCGP08].

There are many different ways to define web services [ACKM04][RS05]. In practice each of the available web services is described with the language WS-BPEL [oas07]. Then the composition problem is:

given a set of available services and a target specification, synthesize an orchestrator that combines the services in such a way as to meet the specification.

In order to tackle the problem of orchestration, researchers use a more formal and abstract representation of web services than WS-BPEL documents. In most work on web service composition, services are represented as a finite state machines with a varying degree of sophistication. Some approaches use state machines that can perform communication actions only [OAS$^+$12], internal action only [BCGP08], internal actions are unobservable [BPT10], internal and communication actions [BCDG$^+$05a] or communication actions indirectly via an environment [GPS13].

1.2 Our approach

Our approach consists of following components:

- A set of available services
- A target or goal service
- An environment
- An orchestrator

The available and target services as well as the environment are modelled as labelled transitions systems. The target service describes a certain behavior that one must satisfy using the available services. In other words, the target plays the role of a specification. The environment represent everything that is not modelled by the services themselves. Constraints on the actions of services can be imposed using the environment. For example a service cannot make a certain transition unless the environment is in particular set of states. The environment is also driven by the services, it changes states depending on the actions of the services. This behavior of the environment allows us to model the input and output of the services indirectly. In other words, the services communicate via the environment. Each available service can be perform a set of actions, some of them controllable and other are uncontrollable. The orchestrator communicates with the available services and makes them peform actions. The behavior composition problem is this

Given a set of available services, a target service and a description of the environment, synthesize an orchestrator such that the synthesized action of the available services in the presence of the environment is equivalent to the target service

The exact definition of "behavior equivalence" can vary from one setting to another. In our approach we use the behavior equivalence used in the Roman Model, namely that for any history of the system the orchestrated service can perform any action that the target can perform. This definition turns out to be the same as bisimulation equivalence. In terms of target specification, we extended the Roman Model by using modal specifications for the target. The expressiveness of modal specifications make them a real advantage especially when the target is given as a set of specifications. Furthermore, other equivalences, such as bisimulation and simulation equivalences, are special cases of modal specifications.

1.3 Thesis objectives and contributions

This thesis deals with the behavior composition problem. Even though we position our work in the context web services, a behavior can describe the logic and interaction of any component, such as devices or agents, in addition to web services. In fact part of this work was used to solve the behavior problem in multi-agent setting [FF14].

Usually synthesis is an automated construction of a system from scratch to conform to a certain specification. In our case we want to synthesis from a set of already existing components, which could be web services or agents.

When tackling the composition problem, a major concern has been the large state space of typical situations which made existing approaches very compute intensive. The aim of this thesis is to extend the expressive, or modeling, power of existing models as well as develop efficient methods to solve the behavior composition problem. Our contributions in that regard are:

1. We formulated a model that takes into account uncontrollable actions by the services. This model is studied for the case when the orchestrator had *perfect* or *partial* information. In both cases we developed a necessary and sufficient condition for the existence of the solution in terms of a controllability relation, for the case of perfect information, and a set of observable relations, in the case of partial information. Such characterization allows us to devise efficient algorithms for the orchestrator synthesis. We showed how an orchestrator can be synthesised once the relations are found.

2. We went beyond existing approaches and used the very expressive *modal specification* to model the target, or required, behavior. A goal for the composition expressed using modal specification is essentially a *set of* acceptable behaviors.

3. We developed an on-the-fly algorithm for the case of perfect information to compute the controllability relation. The importance of such an algorithm lies in the fact that it, unlike other approaches, does not need to visit all the states of the system which are typically prohibitively large. Another advantage, which is not present in other approaches, is the possibility of using heuristic to reduce the search even further. We also proved that our proposed algorithm is robust to component failure, in the sense that if a component fails the re-computation does not need to restart from scratch, but reuses the information collected before the failure.

4. We propose a heuristic to be used with the above mentioned algorithm, based on an abstraction method that reduces the size of the state space drastically. Such an abstraction allow us to infer the non-existence of a solution to the original problem from the non-existence of a solution to the much smaller abstracted problem. Furthermore, if a solution to the abstracted problem exists it is used as an input to the on-the-fly algorithm to speed up the search for a solution to the original problem.

5. We developed a on-the-fly algorithm for the case of partial information. Here the problem would be 2EXPTIME. In other approaches one has to determinize the labelled transition system before computing a solution, even if no solution exists. in our algorithm this determinization is done on-the-fly while finding a solution. When a solution does not exist this will speed up matters considerably

1.4 Related work

As mentioned in the previous section, there are many widely different approaches to the composition problem. In this section we discuss the ones that are similar to ours.

1.4.1 Work of Balbiani et. al.

One approach that is closely related to, and inspired, our work is by Balbiani, Cheikh and Feuillade [BCF08a][BCF09]. Their approach models the composition problem using the following components:

- A client service that represents all the possible communications with a potential client.
- A goal service that communicate with the client service.
- A community of available services.
- An orchestrator (or mediator) service.

The above services are represented with communicating automata, where each automaton has ports that can hold a finite number of messages. In this model the behavior composition problem is this:

Synthesize an orchestrator such that the available services under orchestration is equivalent to the target service.

The equivalence considered are trace, simulation or bisimulation equivalence. In [BCF08a] they showed that their model is equivalent to the classical controller synthesis problem [RW89]. They proposed a method to solve it by reducing the problem to the satisfiability of a μ-calculus formula. This reduction is then amenable to a solution using the method of Arnold. et. al. [AVW03]. Another method to solve the case of asynchronous communication is via a filtration technique [BCF10].

The aforementioned work deals with automata that communicate asynchronously. The work of Balbiani, Alili, Héam, and Kouchnarenko [BAHK10], considered the problem of synchronous communication. This means that a service cannot receive a message unless another is sending and vice versa. Also in [BAHK10] conditional actions are considered where an action is executed only if its precondition is satisfied.

It should be noted that in all the above work the services are considered to be partially controllable and partially observable. The only controllable actions are the communication actions between the orchestrator and the services. In this sense the controller can control the services only by sending them messages. All other, internal, actions are uncontrollable.

1.4.2 Work of Bertoli et. al.

An approach that is similar to ours is the work of Bertoli, Pistore, and Traverso, which started in [PBB+04][TP04] and culminated in [BPT10]. This approach uses the planning via model-checking approach developed in [PT01][BCPT03].

The authors reduce the web service composition problem to a planning problem where the planning domain is the asynchronous product of available services. In this approach, web services whose behavior is published as an BPEL specification, are converted to state transitions systems (STS). This conversion entails the division of the STS actions into two groups: input/output and internal actions. As the name suggests, the input/output actions are where the services can communicate with external environment (i.e. controller) and the internal actions, all labelled by τ, are considered as uncontrollable and unobservable actions. Then the service composition problem is reduced to:

synthesize a controller that interacts with the asynchronous product of services in such a way as to satisfy a goal formula.

To solve the composition problem a *belief* level system is constructed first. A *belief* state, first introduced in [GB96], is basically the set of states that the system can possibly be in, following a given set of observable actions. That it is not a single state is due to the fact that some actions are not observable which means that the system could have evolved without the controller knowing about it. Once this belief level system is constructed, and given a set of services and a logical specification for the goal, they find a solution using an algorithm adapted from [CPRT03]. While the algorithm they use is similar to a reachability game [dAHK07] they provide no analysis of its computational complexity.

Beside the lack of complexity analysis their model is based on a few assumptions/simplifications that reduce the power and generality of their model. This is especially true when the problem to be solved is the general behavior composition and not just the composition of web services. First internal transitions are assumed to be deterministic which means in any given state only a single τ transition exists. Also they assume that the transition system that model services contain no loops which means all histories are finite. It is worth mentioning that they developed an on-the-fly approach to generate the belief state in [BPT06] but there is no complexity analysis of the overall method.

1.4.3 Work of De Giacomo et. al.

Unlike the work of Bertoli et. al., the model of De Giaocomo, Patrizi, and Sardiña [GPS13] considers internal actions as well as communication actions. This approach to the composition problem started in [BCG+03] by Berardi et. al. and extended in [BCDG+05a] by introducing data and communication capabilities. In this extension conditional automata represent OWL-S documents. The resulting model, called COLOMBO, consist basically of the following components:

- A database
- Available services that share the database.
- A client service that can execute communication actions only.
- A goal service that represent the expected behavior from the client.
- An orchestrator or mediator service that represent the service composition.

Given the above components the composition problem is given as:

Synthesize a mediator (orchestrator) service such that the asynchronous product of the client and goal service is isomorphic to the asynchronous product of the meditator with the available services and the client service.

The above isomorphism is modulo the actions of communications. The problem is then solved by reducing it to a satisfiability of a Propositional Dynamic Logic formula (PDL)[BCDG$^+$05a][DGS07]. They also show that, with some constraints on the problem, an upper bound of double exponential. At a later work a slightly simplified version of the problem is solved using the classical concept of simulation where the services are considered as deterministic services [BCGP08]. The formalism was later extended to non-deterministic services using the concept of Non-deterministic simulation (ND-simulation) [SPD08]. A comprehensive summary of the approach is given in [GPS13]. In this latest version the community of available services in the presence of an environment is controlled by an orchestrator. The behavior composition is defined as:

Synthesize an orchestrator such that the behavior of the asynchronous product of services as controlled by the orchestrator is behavior equivalent to the target service.

The equivalence of behavior means that for all possible histories of the community of service and the target, any action performed by the target can also be performed by the community under orchestration.

It should be noted that in this approach, which will be referred to henceforth as the Roman Model, partial controllability is introduced via non-determinism only. Unlike the standard approach to controller synthesis, in the Roman Model all actions are considered to be controllable. Therefore the behavior they simulate using non-determinism is more appropriately called partially predictable rather than partially controllable.

In the above references the available services are considered as fully observable. Services with partial observations were tackled in [GMP09]. The model in [GMP09] assigns observation to states according to some observation function but, unlike the classical controller synthesis problem, all actions are considered to be observable. Therefore one can distinguish between two seemingly indistinguishable states by observing the sequence of actions executed to reach them. Unless these two states can be reached by the same sequence of actions they will be distinguishable.

The case when the system is partially observable was studied in [GMP09] by reducing the problem to a fully observable system using the concept of belief state [GB96]. Basically a belief state is a set of all the states that the system could potentially be in. Once this reduction is achieved the same method as in the case of full observation is used.

When the system is fully observable this group proposes different methods to synthesize an orchestrator. Initially they showed[SPD08] that an orchestrator can be synthesized from a variant of the simulation relation [Mil71] called non-deterministic simulation (ND-simulation). The ND-simulation is obtained in the standard fashion as the largest fix-point of a function over the relation space. Other approaches taken by the group was to reduce the problem to safety games [DGP10] and Alternating Temporal Logic (ATL) model checking [DGF10].

1.4.4 Work of Oster et. al.

In the approach of Oster, Ali, Santhanam, Basu and Roop [OAS+12] the authors provide a unified framework for functional and non-funcitonal specification as well as behavior constraints. They model the services by Labelled Transition Systems (LTS) that can perform input/output actions only, without internal actions. Furthermore, a set of propositions is associated with each state. They provide a comprehensive framework that can handle required and optional specifications. The required specification is further divided into functional requirements, meaning *what* capabilities the orchestrated system should provide. These functional requirements are specified as an AND-OR graph. This representation of the requirement has greater flexibility since a requirement can be decomposed into different atomic requirements [OSB11]. The second component of the required specification is called behavior constraint and handles *how* the functionality is provided. These behavior constraint are specified using CTL. The user supplied non-functional requirements are specified in language called Conditional Importance Networks (CI-nets)[BEL09]. These networks define a sort of preorder over the preferences such that one can quantify the statement "one solution is preferable to another solution".

In their model a composition is the synchronous product of all Labelled Transition Systems where the output of one LTS is consumed by another. The behavior composition problem in this case is:

> *Find a composition that satisfies the functional requirements and constraints such that it is preferable to any other composition satisfying the requirements and constraints*

1.4.5 Comparison with our work

The model we use in this thesis is very similar to the one by Balbiani et.al. and and De Giacomo et. al. Our model is slightly less expressive than the Balbiani group but more expressive than the De Giacomo group. In the approach Balbiani et. al. they deal explicitly with input/output actions and handle them differently from internal actions. In our approach the input/output actions are handled implicitly by embedding them in the environment as is done in the approach of giacomo. Unlike the De Giacomo group we handle uncontrollable actions as well as controllable

actions as done in the approach of the Balbiani group. Furthermore, the representation of the goal in our case is similar to both groups but we go further by using modal specification for the target. In fact our approach is inspired by and we regard it as a continuation of the work of both groups.

The models of Oster et. al. [OAS+12] and and Bertoli et.al. [BPT10] are further away from ours. First their goal is for the orchestrated system to satisfy a logical formula and not to be equivalent to some target behavior as it is in our model. While formally the two approaches are equivalent in practice the comparison between the two is much more difficult. Also in Oster et. al. there are no internal transitions where as in our model internal actions are an essential component. When one deals with behavior composing other than web services, for example multi-agent systems, non-communication actions are essential. The model of Bertoli et. al. goes a bit further by abstracting all internal actions as τ transitions which makes them unobservable. In fact as far as the internal actions are concerned, Bertoli's model becomes a special case of our model (the case of partial information) by reducing the set of unobservable actions to one, τ.

Regardless of the model and especially in the case of partial information the behavior composition has high complexity as a function of the number of services, EXPTIME in the perfect information case and 2EXPTIME in the partial information case. None of above approaches deals with this high complexity, rather they rely on standard tools, mainly OBDD techniques, to solve the problem. A novelty in our approach is the use of an on-the-fly algorithm both for the case of perfect and partial information. The use of these algorithms in conjunction with the proposed abstraction method as a heuristic offer a promising approach to a problem with such high complexity.

1.5 Organization of the thesis

In chapter 2 we first present the model and the basic definitions. We show that our model extends the Roman Model in two ways. First our model includes uncontrollable actions which are absent from the Roman Model. Second we use modal specifications to model the target behavior. Modal specifications are more general and less restrictive than target specification as used in the Roman Model. Also in chapter 2 we prove that an orchestrator with perfect information exists if and only if there is a *controllability* relation between the community of services and the target. This result is also proved for the case when modal specification is used for the target. Similarly, we prove that an orchestrator with partial information exists if and only if a set of relations, which we call *observation relations*, exists between the community of services and the target. We also extend the aforementioned results to the case when an environment is present. In chapter 3 we give a new on-the-fly algorithm to compute the observability relation in the case of perfect information. The complexity and correctness of the algorithm is proven. We also introduce an abstraction method that is used as a heuristic to speed up the algorithm. The general algorithm which includes modal specification is also given. In chapter 4 the case of the orchestrator with partial information is tackled. We present a new on-the-fly algorithm to compute the set of

observation relations. This algorithm obviates the need for the pre-computation of the subset construction. We prove the correctness and complexity of the algorithm. We conclude and give future directions in chapter 5

Chapter 2

Framework and general results

Contents

2.1 Introduction

This chapter contains the necessary definitions and the formal setting of the behavior composition problem. It also contains original results that are part of the contributions of this thesis to the behavior composition problem.

Before stating the problem formally, it is useful to give an overview of the involved components and how they fit in the framework. In this work we follow a line of reasoning similar to [GPS13], a model originally proposed in [BCG$^+$03], and [BCF08a]. The needed components include: a set of *available services*, a *target service*, and an *environment*.

Each service in the set of *available services*, or *community* of available services, is a component whose behavior is published by a service provide and performs certain tasks by communicating with the client.

The *target service* is a service whose behavior we require. In another words it is a specification of a behavior that one needs to satisfy.

The *environment* is a system shared by all the services, which allows them to maintain state and communicate. The environment also serves as a vehicle to impose behavioral constraints on the actions of some services. For example, in the case of one service needing the output of another service to proceed this is modelled by allowing the transition only when the environment is in a certain state.

Having introduced the components we can state the *behavior composition problem* informally as: *given a target service and a set of available services, find an orchestrator, if one exists, that communicates with the services in such a way as to delegate requested actions to suitably chosen available services, such that the system will have the same behavior as the target service.*

First in section 2.2 we introduce the basic model for the services and service composition that will be used throughout the thesis. In section 2.2.1 we show that the so called Roman Model is a special case of our model. The case of composition with perfect information is discussed in section 2.3. We use the concept of controllability relation, introduced in section 2.3, to present the first contribution of this thesis: we prove that an orchestrator with perfect information exists if and only if a controllability relation exists between the community and the target. Furthermore, in section 2.3.3 we explore the relation between the behavior composition problem and control theory where we show that behavior equivalence is the same as bisimulation equivalence. In section 2.3.4 we use the expressive and less restrictive modal specifications as a behavior specification. To our knowledge modal specifications have not been studied in the context of behavior composition before. We prove that in the case of modal specifications the existence of an orchestrator depends on the existence of a controllability relation. Then the case of composition with partial information is presented in section 2.4. In that section we introduce the concept of a set of *observation relations* and use it to present another contribution of this thesis: we prove that an orchestrator with partial information exists if and only if there exists a set of observation relations between the community and the target. In both the perfect and partial information case, we show how to construct the orchestrator from the controllability relation and set of observation relations respectively. Since communicating services can be modelled by

an environment, the environment in introduced in section 2.5 where the same results obtained without the presence of the environment are shown to be also valid in the presence of the environment. The conclusion is given in 2.6.

2.2 A model of services

In our framework services are modelled as labelled transition systems (LTS). The transitions of a given service can be divided into two categories: controllable and uncontrollable. The former can be enabled/disabled by an *orchestrator* whereas the latter are "spontaneous" transitions that cannot be controlled. The system under study consists of three components:

1. A community of n available services.
2. A target or goal service whose behavior we need to mimic.
3. An orchestrator that communicates with the available services to "realize" the behavior of the target service.

The basic ideas is to find, or more accurately synthesize, an orchestrator that can modify the behavior of the available services in such a way as to be satisfy the goal. Having introduced the major components intuitively we give next a more precise and formal definitions. We start with available services.

Definition 2.2.1 (Available Service)**.** *An available service $\mathcal{S}_i$ is a tuple $\mathcal{S}_i = \langle S_i, \Sigma_i, Com_i, s_i^0, \delta_i \rangle$ where*

- *S_i is a finite set of states.*
- *Σ_i is a finite action alphabet.*
- *Com_i is a finite set of communication messages.*
- *s_i^0 is the initial state.*
- *$\delta_i \subseteq S_i \times (\Sigma_i \cup Com_i \times \Sigma_i) \times S_i$ is the transition relation.*

For $s, s' \in S_i$ and $a \in \Sigma_i$ it is sometimes convenient to write $(s, a, s') \in \delta_i$ as $s \xrightarrow{a} s'$. Similarly, for $m \in Com_i, a \in \Sigma_i$ it is sometimes convenient to write $(s, \{m, a\}, s')$ as $s \xrightarrow{m|a} s'$. Also, one can use a functional notation for δ_i, such that $\delta_i(s, a)$ denote the set of states $\{s' \mid s \xrightarrow{a} s'\}$. An LTS is called *deterministic* if the set $\delta_i(s, \alpha)$ has at most single state for all $s \in S_i$ and $\alpha \in \Sigma_i \cup Com_i \times \Sigma_i$.

The conditional notation $s \xrightarrow{m|a} s'$ means that the service performs action a only if it receives a message m. More precisely, the action a is performed if it is *enabled* by the orchestrator. We call such a transition a *controllable* transition because whether it happens or not depends on the

orchestrator. The *uncontrollable* actions are all the actions without a condition, e.g. of the form $s \xrightarrow{a} s'$ where $a \in \Sigma_i$.

This situation is similar to the case in control theory where some actions are controllable and others are not. One can regard the present situation as dual to the classical control problem. In the classical problem controllable actions can be performed unless explicitly *disabled* by the controller. In our case the controllable actions are performed if they are *enabled* by the orchestrator. Obviously the orchestrator can disable an action, say $s \xrightarrow{m|a} s'$, like a classical controller by *not* sending a message m.

The potential of behavior composition stems from the fact that there are many services or components that can be orchestrated. Such a collection of services is referred to in this thesis as a *community of available services*. It is essentially an asynchronous product of n available services. The formal definition is given as:

Definition 2.2.2 (Community of Services)**.** *A community of n available services $\mathcal{S}_i = \langle S_i, \Sigma_i, Com_i, s_i^0, \delta_i \rangle$, $i = 1 \ldots n$, is the tuple $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ where*

- $S = S_1 \times \ldots \times S_n$.
- $s^0 = (s_1^0, \ldots s_n^0)$.
- $\Sigma_u = \cup_i \Sigma_i$
- $Com = \cup_i Com_i$
- $\delta_u \subseteq S \times (\Sigma \cup Com \times \Sigma) \times S$

The transition relation δ_u is the asynchronous product of all relations δ_i defined as

$$(\langle s_1, \ldots, s_n \rangle, \alpha, \langle s_1', \ldots, s_n' \rangle) \in \delta_u \text{ iff } (s_k, \alpha, s_k') \in \delta_k \text{ for some } 1 \leq k \leq n \text{ and for all } i \neq k \text{ we have } s_i = s_i'$$

The functional notation δ_u defines the set of tuples:

$$\delta_u(s_1, \ldots, s_n, \alpha) = \bigcup_{i=1}^{n} \bigcup_{s_i' \in \delta_k(s_i, \alpha)} (s_1, \ldots, s_i', \ldots, s_n)$$

Since most of the time we will be discussing the evolution of the system as a whole we use a single state s to mean the set of states of the n services, $s = \langle s_1, \ldots, s_n \rangle$. When we need to label a state of the whole community we use a superscript rather than a subscript which denotes a particular service. For example, two different states of the whole community will be referred to as s^1 and s^2 rather than s_1 and s_2 which refer to states for service one and two respectively. Using this notation for n services, the initial state of service i would be written s_i^0 and the initial state of the community would be written as $s^0 = \langle s_1^0, \ldots, s_n^0 \rangle$.

The subscript u is short for *unconstrained* transition function. Its meaning will become clear when we introduce the orchestrator. Basically it tells us how the system will evolve if *all the actions* were *enabled*. It is useful to divide the *unconstrained* actions into two parts: uncontrollable and controllable. Recall that controllable actions are conditional, i.e. always prefixed with a message. Therefore when we write $\delta_u(s, a)$ we mean the set of states reached by making an *uncontrollable* a-transition from state s. Similarly, $\delta_u(s, m \,|\, a)$ means the set of states reached by making a *controllable* a-transition when the community receives message m or when m is *enabled*.

Example 2.2.1. *An example community of services composed of two simple services $\mathcal{S}_0$ and $\mathcal{S}_1$ is shown in Figure 2.1 below. Note the asynchronous nature of the community. When in state (s_0^0, s_1^0) the system can make an uncontrollable c transition. Also from the initial state two controllable transitions can be performed only if the community receives a message α. Note that when a message α is received in state (s_0^0, s_1^0) the community can make either an a or b transition. We can use Figure 2.1 to give examples (non exhaustive) of the notation we have introduced:*

- $\delta_u((s_0^0, s_1^0), c) = \{(s_0^1, s_0^0)\}$
- $\delta_u((s_0^0, s_1^0), a) = \emptyset$
- $\delta_u((s_0^0, s_1^0), \alpha \mid a) = \{(s_0^2, s_1^0)\}$
- $\delta_u((s_0^0, s_1^0), b) = \emptyset$
- $\delta_u((s_0^0, s_1^0), \alpha \mid b) = \{(s_0^0, s_1^0)\}$

In order to explain how the community is controlled it is necessary to explain the role of the orchestrator. But before giving the formal definition of an orchestrator it is useful to give an intuitive explanation how it modifies the behavior of the community. First, the orchestrator can enable/disable *controllable* transitions only. In example 2.2.1, the orchestrator can enable/disable, for example, the $(s_0^0, s_1^0) \xrightarrow{\alpha|a} (s_0^2, s_1^0)$ transition but *cannot* disable the $(s_0^0, s_1^0) \xrightarrow{c} (s_0^1, s_1^0)$ transition. When an orchestrator interacts with the community we call the resulting behavior the *orchestrated* behavior of the community or the behavior of the community *under orchestration*. As an example, assume that the orchestrator enables α in the initial state and disables all other (controllable) transitions. Then the resulting *orchestrated* behavior of the community would be as shown in Figure 2.2 below. Note that the no transition of the *orchestrated* behavior have a precondition anymore.

So far we have not given a precise definition of an orchestrator nor a precise mechanism in which it interacts with the community. In the literature, and in this thesis, there are two types of orchestrators that are usually considered: orchestrators having perfect or partial information. A discussion of what type of information an orchestrator has access to will be left to later sections. In this section we give a general, and therefore a bit vague, definition of an orchestrator.

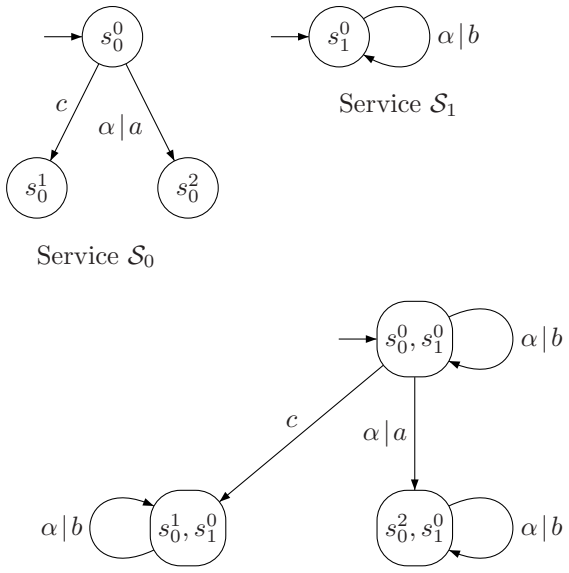


Figure 2.1: Two services $\mathcal{S}_0$ and $\mathcal{S}_1$ forming a community by their asynchronous product $\mathcal{S}_0 \times \mathcal{S}_1$

Definition 2.2.3 (Orchestrator)**.** *Given a community of n services $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$, an orchestrator for that community is a function $\Omega : Com \times D \longrightarrow \{0, 1\}$. Where D is a domain of information that at this point is left unspecified.*

We will see later that D depends on what type of orchestrator we are considering. At this point it is sufficient to say that the action of Ω depends on messages in Com. Therefore in this section we will drop the dependence on D, which is in any case unspecified.

Let $\Omega(m)$, $m \in Com$, denote the action of the orchestrator. Then we can formalize the *orchestrated* behavior (under Ω) of the community by introducing an *orchestrated* transition function δ_Ω. Then $\delta_\Omega(s, a)$ is interpreted as the set of states reached, under orchestration (by orchestrator Ω), from state s when an a-transition is performed. Such an a-transition can be either *uncontrolled* or a *controlled* that was *enabled* by the orchestrator. As mentioned before, in this section the only input parameter for the orchestrator is the message. Also since all uncontrollable actions are from Σ_u and all controllable action are of the form $\alpha \,|\, a$ with $\alpha \in Com$ and $a \in \Sigma_u$ then the *orchestrated* transition function, or the transition function of the *orchestrated community* is written as:

$$\delta_\Omega(s, a) = \delta_u(s, a) \bigcup_m \delta_u(s, m \mid a) \odot \Omega(m)$$

Note that in the above definition the term to the left of the union symbol represents the contribution of the *uncontrollable* transitions whereas the term to the right represents the contribution of

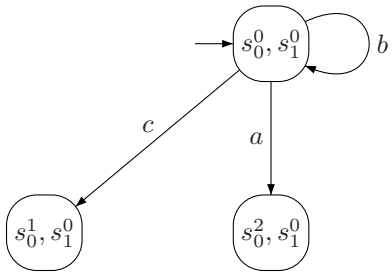


Figure 2.2: The *orchestrated* behavior of the community of Figure 2.1 when the orchestrator disables all controllable actions except the ones with precondition α at the initial state.

the *controllable* transitions. The symbol $\odot$ is a convenient notation to model the on/off behavior of the orchestrator. This operation can be described by its effect as

$$\delta_u(s, m \,|\, a) \odot \Omega(m) = \begin{cases} \emptyset \text{ if } \Omega(m) = 0 \\ \delta_u(s, m \,|\, a) \text{ if } \Omega(m) = 1 \end{cases}$$

It should be emphasized that the *orchestrated* community has no conditional transitions any more, all transitions are labelled from the action alphabet Σ.

Example 2.2.2. *We use Figures 2.1 and 2.2 to give a few examples of the formal notations we just introduced. Since the orchestrator is "on" only at the initial state and only for the message α then we can write:*

$$\begin{aligned} \delta_\Omega((s_0^0, s_1^0), a) &= \delta_u((s_0^0, s_1^0), a) \cup \delta_u((s_0^0, s_1^0), \alpha \mid a) \odot \Omega(\alpha) \\ &= \emptyset \cup \{(s_0^2, s_1^0)\} \\ &= \{(s_0^2, s_1^0)\} \end{aligned}$$

In this example the orchestrator disables the transitions with α precondition in the state (s_0^1, s_1^0) therefore, as an example,

$$\begin{aligned} \delta_\Omega((s_0^1, s_1^0), b) &= \delta_u((s_0^1, s_1^0), b) \cup \delta_u((s_0^1, s_1^0), \alpha \mid b) \odot \Omega(\alpha) \\ &= \emptyset \cup \{(s_0^1, s_1^0)\} \odot 0 \\ &= \emptyset \cup \emptyset \\ &= \emptyset \end{aligned}$$

We will have more to say about the orchestrator when we define the different types. Since the main objective of this thesis is to develop efficient methods for the behavior composition problem. Such a problem entails finding a way to orchestrate the different available services so that they have a behavior equivalent to some target behavior. In our model the target behavior

is represented as a *target service*. There are many ways to describe the behavior of two systems as equivalent. If the behaviors are represented by transitions system then one can use classical equivalences such as trace equivalence or, as is done in this thesis, behavior equivalence. We will come back to behavior equivalence at a later stage, after we introduce all the necessary concepts.

Before proceeding any further we need to give a formal definition of the *target service*, which we denote in this thesis by $\mathcal{S}_t$.

Definition 2.2.4 (Target Service)**.** *A target service $\mathcal{S}_t$ is a tuple $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, \delta_t \rangle$ where*

- *S_t is a finite set of states.*
- *Σ_t is a finite action alphabet.*
- *t^0 is the initial state.*
- *$\delta_t \subseteq S_t \times \Sigma \times S_t$ is the transition relation.*

As seen from the above definition the target service is similar to available services with the exception of being required to be a *deterministic* LTS. As explained before, the target being a deterministic LTS means that for all $a \in \Sigma_t$ and $t \in S_t$ the set $\delta_t(t, a)$ can have at most one state.

To be able to compare the behavior of the orchestrated community and the target it is important to describe how they both evolve over a sequence of actions. Also, it is convenient to describe their behavior with reference to the same action set. For the *orchestrated* community we define the extended function $\Delta_\Omega : S \times \Sigma^* \to 2^S$, where $\Sigma = \Sigma_u \cup \Sigma_t$. Let $a \in \Sigma$ and $x \in \Sigma^*$ be an arbitrary action and sequence of actions respectively, $s \in S$ be an arbitrary state of the *orchestrated* community, then the function Δ is defined recursively, with the help of the empty sequence ϵ, as:

$$\begin{aligned} \Delta_\Omega(s, \epsilon) &= \{s\} \\ \Delta_\Omega(s, xa) &= \bigcup_{s' \in \Delta_\Omega(s,x)} \delta_\Omega(s', a) \end{aligned}$$

We use the same symbol for the extended function for the target

$$\begin{aligned} \Delta_t(t, \epsilon) &= \{t\} \\ \Delta_t(t, xa) &= \bigcup_{t' \in \Delta_t(s,x)} \delta_t(t', a) \end{aligned}$$

It should be noted that the target is deterministic so there is really no need for the union symbol in the above definition but we keep it for consistency.

2.2.1 Roman Model

In this section we give a brief description of the so called Roman Model as used in [BCDG$^+$05b][BCGP08][DGP10 and compare it with the model we have presented in the previous section. In the Roman Model the orchestrator delegates actions to the services. Given a trace x and action a the orchestrator chooses a service k to performs the action. Formally each service can be represented by the tuple $\mathcal{S}_i = \langle S_i, \Sigma_i, s_i^0, \delta_i \rangle$ where the alphabet is Σ_i. The orchestrator is a function $\Omega : S \times \Sigma^* \times \{n\} \times \Sigma \longrightarrow \{0, 1\}$, where $\{n\}$ is the set of numbers from 1 to n. For example $\Omega(s, x, ka) = 1$ means in community state s and after performing the trace x the orchestrator delegates the next action a to service k. It is easy to see that this model is a special case of the more general model presented in the previous section. Below we give the translation from our model to the special case of the Roman Model for the case of an orchestrator with perfect information. The case with partial information is similar. Let $s \in S$ be an arbitrary community state and $\tau \in \Sigma^*$ be a sequence of actions executed to reach s from the initial state s^0. The Roman Model is obtained from our general model as follows:

- Restrict the model by removing the uncontrollable evolution of the system. In this case the single step evolution of the system, which was written as

$$\delta_\Omega(s, a) = \delta_u(s, a) \cup \bigcup_{\alpha \in Com} \delta_u(s, \alpha \,|\, a) \odot \Omega(s, \tau, \alpha)$$

  by removing the uncontrollable component becomes:

$$\delta_\Omega(s, a) = \bigcup_{\alpha \in Com} \delta_u(s, \alpha \,|\, a) \odot \Omega(s, \tau, \alpha)$$

- The possible set of messages that can be received by a service reduces to the form ka where k is the service index and $a \in \Sigma$ is the desired action. Thus the evolution of the system becomes:

$$\delta_\Omega(s, a) = \bigcup_{i \in \{1, \ldots n\}} \delta_u(s, i \,|\, a) \odot \Omega(s, \tau, ia)$$

It is clear from the above that the basic Roman Model is too restrictive. First, all actions of the services are assumed to be controllable which is not always the case. Furthermore, the orchestrator has full control over the choice of action of the services without any side effects. In our model, a message α sent by the orchestrator to the community can enable a desired action, say a, but it can also unintentionally activate another action b. This is totally absent from the Roman Model since the orchestrator can specify exactly which action to enable. Also in our model the orchestrator does not specify which service to perform a given action. If the orchestrator sends a message α to the community it has no control over which actual service reacts to the

message. An example from real life would be the case of a building having multiple elevators and by pressing the button we don't actually know which elevator responds. In summary, our model is a much needed enhancement over the basic Roman Model.

In many other approaches to synthesis problems in different fields, two types of orchestrators/controllers are considered: those having perfect information and those having partial information. In this thesis we also study both types. When an orchestrator with perfect information is considered we assume that it has full knowledge of which state the community is in and the sequence of actions the community performed to reach that state. When an orchestrator with partial information is considered we assume, as in control theory, that the actions of the community are divided into observable (by the orchestrator) and non-observable actions. In our model, unlike control theory, the set of controllable (uncontrollable) actions is the same as the set of observable (unobservable) actions. The reason is that the orchestrator is aware only if its own actions, which are the messages it sends to the community. In the next section we study the composition when the orchestrator has perfect information and in section 2.4 we study the case with partial information.

2.3 Composition with perfect information

In this section we study the behavior composition problem when the orchestrator has perfect information. Having perfect information means that, at any point, the orchestrator knows the state in which the community is in, and the sequence of action performed by the community to reach that state. The orchestrator can then use that information to decide whether to enable or disable certain controllable transitions.

Now we can refine definition 2.2.3 by identifying the domain D, which was left unspecified, as $S \times \Sigma^*$. The complete definition follows.

Definition 2.3.1 (Orchestrator with perfect information)**.** *Given a community of n services, $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$, an orchestrator with perfect information is a function $\Omega : S \times \Sigma^* \times Com \longrightarrow \{0, 1\}$.*

The values 1 and 0 denote the *enabling/disabling* of a particular message. For example, let $s \in S$, $w \in \Sigma^*$ and $\alpha \in Com$ then $\Omega(s, w, \alpha) = 1$ means that if the community reached state s after a sequence of actions w, the orchestrator *enables* all actions prefixed by the message α.

Note that $\Sigma = \Sigma_u \cup \Sigma_t$. We could have use Σ_u only, but it is more convenient to use an action set common to both the community and the target. Obviously the community cannot perform actions in $a \in \Sigma \setminus \Sigma_u$ so in that case for all $s \in S$ we have $\delta_u(s, a) = \emptyset$ and for all $m \in Com$, $\delta_u(s, m \mid a) = \emptyset$. Similarly for the target, for all $a \in \Sigma \setminus \Sigma_t$, $\delta_t(t, a) = \emptyset$.

Having a more precise meaning of the orchestrator we refine the definition of the *orchestrated* transition function

$$\delta_\Omega(s,a) = \delta_u(s,a) \cup \left[\bigcup_{m \in Com} \delta_u(s, m \,|\, a) \odot \Omega(s, Tr(s), m) \right] \tag{2.1}$$

Where $Tr(s) \in \Sigma^*$ is the sequence of actions performed by the community to reach state s. Basically an orchestrator sends a message to services based on the current community state *and* the sequence of actions (i.e. trace) executed to arrive at that state.

The above definition as presented poses some questions. First, a state can be reached by many different paths therefore $Tr(s)$ is not unique. As we will see later, only the sequence of actions is needed and not the exact path taken. This is because the target service is deterministic and for a given trace there is exactly one target path. In other words, ultimately the dependence of the orchestrator will be on the community *and* target state: $\Omega = \Omega(s,t,m)$ where t is some target state. Second, as we will see later the expression $\Omega(s, Tr(s), m)$ is never used by itself but occurs in a context of a history of actions and in that case the computation of $Tr(s)$ is actually unique. This value will be clearer once we define how the community evolves after a sequence of actions.

Example 2.3.1. *Using the example in Figure 2.1 we define the orchestrator's values as*

$$\Omega((s_0^0, s_1^0), \epsilon, m) = \begin{cases} 1 \text{ if } m = \alpha \\ 0 \text{ otherwise} \end{cases}$$

Note that in this case the sequence of actions executed to reach the starting state (s_0^0, s_1^0) is the empty sequence ϵ. The evolution of the community on action a would be:

$$\begin{aligned} \delta_\Omega((s_0^0, s_1^0), a) &= \delta_u((s_0^0, s_1^0), a) \cup \bigcup_{m \in Com} \delta_u((s^0, s_1^0), m \,|\, a) \odot \Omega((s^0, s_1^0), \epsilon, m) \\ &= \emptyset \cup \delta_u((s_0^0, s_1^0), \alpha \,|\, a) \\ &= \emptyset \cup \{(s_0^2, s_1^0)\} \\ &= \{(s_0^2, s_1^0)\} \end{aligned}$$

similarly for action b:

$$\begin{aligned} \delta_\Omega((s_0^0, s_1^0, b) &= \delta_u((s_0^0, s_1^0), b) \cup \bigcup_{m \in Com} \delta_u((s_0^0, s_1^0), m \,|\, b) \odot \Omega((s_0^0, s_1^0), \epsilon, m) \\ &= \emptyset \cup \delta_u((s_0^0, s_1^0), \alpha \,|\, b) \\ &= \emptyset \cup \{(s_0^0, s_1^0)\} \\ &= \{(s_0^0, s_1^0)\} \end{aligned}$$

It is important to realize that the orchestrator cannot enable the $(s_0^0, s_1^0) \xrightarrow{\alpha|a}$ actions without enabling the $(s_0^0, s_1^0) \xrightarrow{\alpha|b}$ action. In this sense the orchestrator does not have *full* control even

on the *controllable actions*. One can consider the special case where the orchestrator's decision depends on the next action also, in our example $\Omega = \Omega(s^0, \epsilon, m, a)$. This condition of *total* control is assumed by the Roman Model.

The original community had conditional a and b transitions whereas the *orchestrated* community has a and b transitions without conditions. The fact that $\delta_\Omega((s_0^0, s_1^0), a) = \{(s_0^2, s_1^0)\}$ can be also written as $(s_0^0, s_1^0) \xrightarrow{a} (s_0^2, s_1^0)$, i.e. with no precondition. The above motivates the definition of the formal definition of the *orchestrated* community.

Definition 2.3.2 (Orchestrated Community)**.** *Let $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ be a community of n services, $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, \delta_t \rangle$ a target service and Ω be an orchestrator with perfect information then the orchestrated community is defined as $\mathcal{S}_\Omega = \langle S, \Sigma, s^0, \delta_\Omega \rangle$.*

The dependence of δ on Ω in the above definition is essential since transitions of a given state can change depending on *how* the state was reached as can be seen from the definition of the orchestrated transition function in equation (2.1). We will defer the construction of $\mathcal{S}_\Omega$ to a later stage. In fact, the construction of $\mathcal{S}_\Omega$ is not necessary as long as one can construct an orchestrator that modifies the behavior of the community in such a ways as to satisfy the target behavior.

Having described the *single step* evolution of the *orchestrated* community under perfect information next we describe the evolution of the *orchestrated* community after a sequence of actions. As before we will use the extended function over arbitrary sequences of actions, i.e. traces. For any trace $x \in \Sigma^*$ and action $a \in \Sigma$ the extended function $\Delta_\Omega : S \times \Sigma^* \to 2^S$ is defined recursively:

$$\Delta_\Omega(s, \epsilon) = \{s\}$$

$$\Delta_\Omega(s, xa) = \bigcup_{s' \in \Delta_\Omega(s,x)} \delta_u(s', a) \cup \left[\bigcup_{m \in Com} \delta_u(s, m \,|\, a) \odot \Omega(s, x, m) \right]$$

The above definition clarifies the exact meaning of $Tr(s)$ term that was ambiguous when the single step transition function was considered. Having described the evolution of the target and the community, in the next section we present the behavior composition problem.

2.3.1 Service composition

The behavior composition problem is the ability of finding an orchestrator that controls or delegates actions to different available services in such a way as to *realize* or *mimic* the behavior of a target service. Formally,

Definition 2.3.3 (Behavior Composition With Perfect Information)**.** *Let $\mathcal{S}_t$ be a deterministic target service and $\mathcal{S}$ be community of n available services. Let Ω be an orchestrator with perfect information and denote by $\mathcal{S}_\Omega$ the orchestrated community. We say that $\mathcal{S}_\Omega$ is a behavior composition of $\mathcal{S}_t$ iff for all traces $\tau \in \Sigma^*$ and all target states $t \in \Delta_t(t^0, \tau)$ and for all $s \in \Delta_\Omega(s^0, \tau)$ we have*

$$\forall a \in \Sigma, \quad \delta_t(t, a) \neq \emptyset \Leftrightarrow \delta_\Omega(s, a) \neq \emptyset$$

The above means that for any possible history of the target after performing a sequence of actions τ, no matter in which state the community is in (due to non-determinism) if the target service can make an a transition then the community, with the help of the orchestrator, will be able to make the same transition regardless in which actual state the community is in. Since the target is deterministic then after a sequence of actions τ there is a single possible state it could be in, but we retain this general definition as it will be useful in later sections. As for the community, because of its non-deterministic transition due to partial controllability it has possibly different set of states it could be in after performing the sequence of actions τ. Then the *orchestrated* community is a composition for the target if no matter in which state it ends up in it will be able to perform any action that the target can perform. Note that the equivalence symbol $\Leftrightarrow$ means that any action not performed by the target should not be performed by the community.

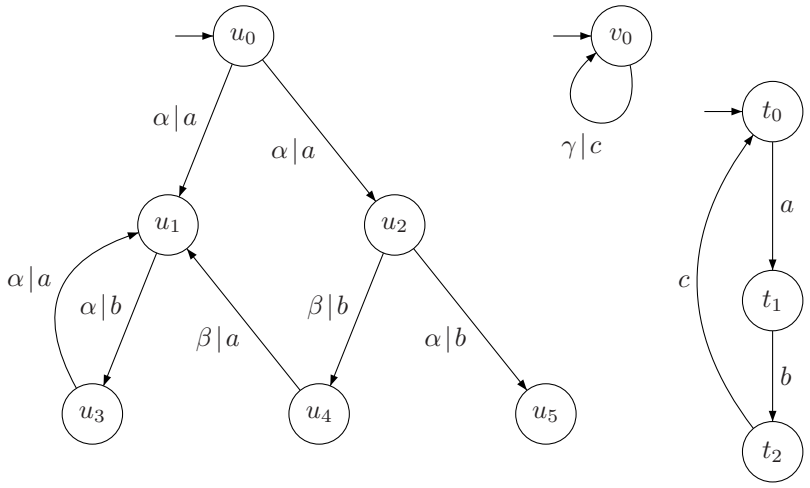


Figure 2.3: Example Composition

Example 2.3.2. *An example composition is shown in Figure 2.3. There are two services U and V and a target service T. To be able to discuss the fine points of the example solution we present it in two parts. First, for an arbitrary trace $x \in \Sigma^*$ a solution to the above problem that deals with the states (u_0, v_0), (u_1, v_0) and (u_2, v_0) involves the following assignments:*

$$\Omega((u_0, v_0), \epsilon, \alpha) = 1$$
$$\Omega((u_1, v_0), xa, \alpha) = 1$$
$$\Omega((u_2, v_0), xa, \beta) = 1$$

First observe the effect of full observability on the orchestrator. The orchestrator can observe in which state the community is in, if it is in state (u_1, v_0) and the required action is b (regardless

of history x) then it sends a message α (or enables action prefixed by α). By contrast if the community is in state (u_2, v_0) then the message sent is β even though the action to be enabled is the same, b. It is easy to see that there is no solution without complete observability since sending message α for both states, leads from state u_2 to the sink state u_5.

The second part deals with the states (u_3, v_0) and (u_4, v_0). These states illustrate the dependence of the orchestrator actions on the whole (or at least part of) history.

$$\begin{aligned}
&\Omega((u_3, v_0), xb, \alpha) = 0 && \Omega((u_4, v_0), xb, \beta) = 0\\
&\Omega((u_3, v_0), xb, \gamma) = 1 && \Omega((u_4, v_0), xb, \gamma) = 1\\
&\Omega((u_3, v_0), xc, \alpha) = 1 && \Omega((u_4, v_0), xc, \beta) = 1\\
&\Omega((u_3, v_0), xc, \gamma) = 0 && \Omega((u_3, v_0), xc, \gamma) = 0
\end{aligned}$$

The above rules for the orchestrator are interpreted as follows. When in state (u_3, v_0) the action c has to be performed before action a is performed. Due to the asynchronous nature of the composite system it is possible to execute an ac or ca sequence and both would be valid if the orchestrator decision depends on the current system state only. Inspecting the target execution, however, informs us that a b transition should be followed by a c transition not an a transition.

This last observation gives us a hint on how to characterize the orchestrator by observing the states of the *target*. If we go through the previous discussion by keeping track of the *target* state we see that when in state (u_3, v_0) and the target is in state t_0 then the orchestrator should enable the a action whereas if the target is in the t_2 state then orchestrator should enable the c action. Since the target is deterministic then there is a one to one correspondence between the traces of the target (and hence the community) and the states of the target. This correspondence allows us to synthesis an orchestrator without "remembering" all the traces, rather by inspecting the target state. This key fact is used in control theory using the concept of controllability.

2.3.2 Controllability

In this section the concept of a controllability relation, or controllability for short, is given and its relation to the behavior composition problem is discussed. The definition presented below is an adaptation of the concept of controllability to the problems we tackle in this thesis.

Definition 2.3.4 (Controllability)**.** *Let $\mathcal{S}_t = \langle S_t, \Sigma_t, s_t^0, \delta_t \rangle$ be a target service and $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ a community of services. A relation $R \subseteq S_t \times S$ is said to be an controllability relation, if for all $(t, s) \in R$ and for all $a \in \Sigma = \Sigma_u \cup \Sigma_t$:*

1. *If $\delta_u(s, a) \neq \emptyset$ then $\exists t'.t \xrightarrow{a} t'$ and for all $s' \in \delta_u(s, a)$ we have $(t', s') \in R$.*
2. *$\exists E(s, t) \subseteq Com$ such that for all $\alpha \in E(s, t)$ we have $s \xrightarrow{\alpha|a} s' \Rightarrow \exists t'.t \xrightarrow{a} t' \wedge (t', s') \in R$.*
3. *If $t \xrightarrow{a} t'$ then:*

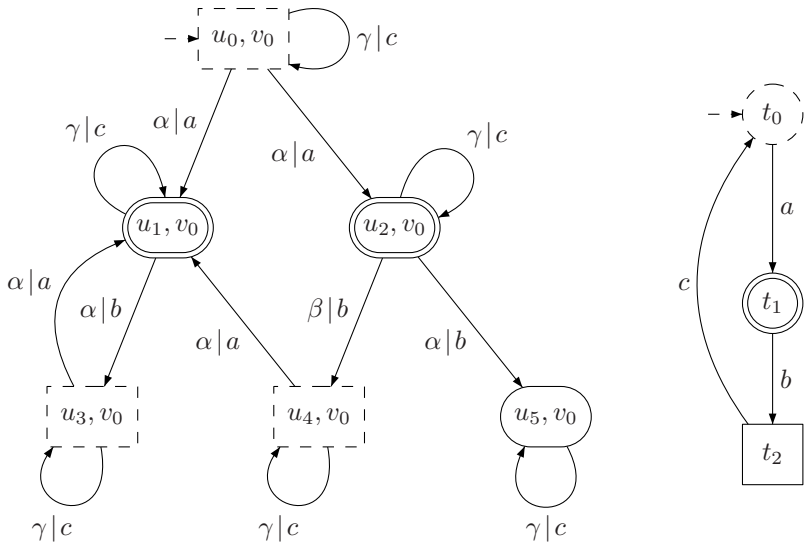


Figure 2.4: Example controllability relation

(a) Either $\delta_u(s, a) \neq \emptyset$ *and for all* $s' \in \delta_u(s, a)$ *we have* $(t', s') \in R$.

(b) Or $\exists \alpha \in E(s, t).\delta_u(s, \alpha \mid a) \neq \emptyset$ *and for all* $s' \in \delta_u(s, \alpha \mid a)$ *we have* $(t', s') \in R$

What is important is to have controllability starting from the initial state.

Definition 2.3.5. *A community of services,* $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ *is said to be controllable with respect to a target service* $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, \delta_t \rangle$ *iff there exists a controllability relation* R *between* $\mathcal{S}$ *and* $\mathcal{S}_t$ *such that* $(s^0, t^0) \in R$.

The controllability relation corresponding to the solution of the example in Figure 2.3 is shown in Figure 2.4. The dashed nodes in the community are related to t_0, the double circled nodes are related to t_1 and the square nodes are related to t_2. Note that some nodes (e.g. (u_0, v_0)) are both dashed and squared and thus related to both t_0 and t_2. The properties of a controllability relation allows one to build an orchestrator if the "proofs" of the controllability relation are saved. For example, (u_0, v_0) is related to t_0 and the "proof" is that the $t_0 \xrightarrow{a} t_1$ transition can be matched by the orchestrator by enabling the transitions prefixed with α. If that information is saved and at some point the target is in state t_0, the community is in state (u_0, v_0) and the target makes an a transition then the orchestrator can determine that it has to enable the actions prefixed by α merely by looking it up in the precomputed controllability relation. This information is saved in a graph representing the controllability and its "proofs" where a node in the graph represents a target state-community state pair and the transitions represent the "proofs". An example of such graph for the controllability in Figure 2.4 is shown in Figure 2.5. It is important to note that even though the orchestrator is related to the controllability graph, the orchestrator itself (for

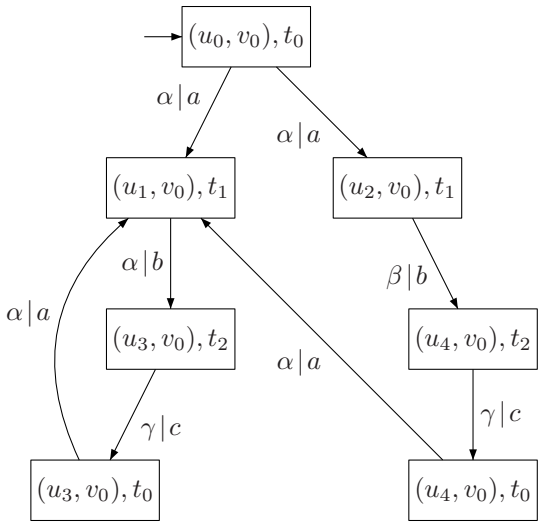


Figure 2.5: Example controllability Graph Corresponding to the controllability relation shown in Figure 2.4

the case of perfect observation) cannot always be represented as an LTS. Rather the orchestrator values can be extracted from the graph as follows:

- Let $\Omega = \Omega(s, \tau, \alpha)$ be the orchestrator we are constructing for every value of the community state s, a trace τ and a message α. We are interested in the values of (s, τ, α) in which $\Omega(s, \tau, \alpha) = 1$, all other values are implicitly assumed to be zero.
- Since the target is deterministic then every trace τ uniquely determines a target state $t = \Delta(t^0, \tau)$. Then
- $\Omega(s, \tau, \alpha) = 1$ iff the node of the controllability graph whose label is $(s, \Delta(t^0, \tau))$ has an outgoing edge labeled α.

As an example, from Figure 2.5 one can deduce that $\Omega((u_0, v_0), \epsilon, \alpha) = 1$ because $\Delta(t^0, \epsilon) = t^0$, and the node whose label is $((u_0, v_0), t_0)$ has an outgoing edge labelled α. The remaining non-zero values of the orchestrator and the corresponding values in the graph are shown in table 2.1.

The previous example was used for its simplicity. Next we introduce a slightly more complicated example to be able to illustrate the finer points of the problem. The second example admits more than one solution. Figure 2.6 shows the same target service as before with two slightly modified community services. The corresponding controllability relation is shown in Figure 2.7. One can see that there are 4 community states that can simulate all 3 target states and they are shown as double rectangles with one of them dashed. The remaining two states of the community can each simulate only two states of the target and they are shown with a single dashed rectangle. What is shown in Figure 2.7 is actually the **largest** controllability between the

Orchestrator Values	Nodes in graph
$\Omega((u_0, v_0), \epsilon, \alpha) = 1$	(u_0, v_0, t_0)
$\Omega((u_1, v_0), xa, \alpha) = 1$	(u_1, v_0, t_1)
$\Omega((u_3, v_0), xb, \gamma) = 1$	(u_3, v_0, t_2)
$\Omega((u_3, v_0), xc, \alpha) = 1$	(u_3, v_0, t_3)
$\Omega((u_2, v_0), xa, \beta) = 1$	(u_2, v_0, t_1)
$\Omega((u_4, v_0), xb, \gamma) = 1$	(u_4, v_0, t_2)
$\Omega((u_4, v_0), xc, \alpha) = 1$	(u_4, v_0, t_0)

Table 2.1: Non-zero Orchestrator values and the corresponding controllability graph nodes.

community and the target. Since the union of two controllability is also an controllability then the largest relation is the union of all controllability. In fact, in this example there are **many** different (elementary) controllability, namely any relation between the states of the target and the states of the community is a controllability as long as it contains one of the following two relations:

$$R_1 = \{[u_0, v_0], t_0), ([u_1, v_0], t_1), ([u_1, v_1], t_2), ([u_1, v_0], t_0)\}$$
$$R_2 = \{([u_0, v_0], t_0), ([u_1, v_0], t_1), ([u_2, v_0], t_2), ([u_2, v_0], t_0)\}$$

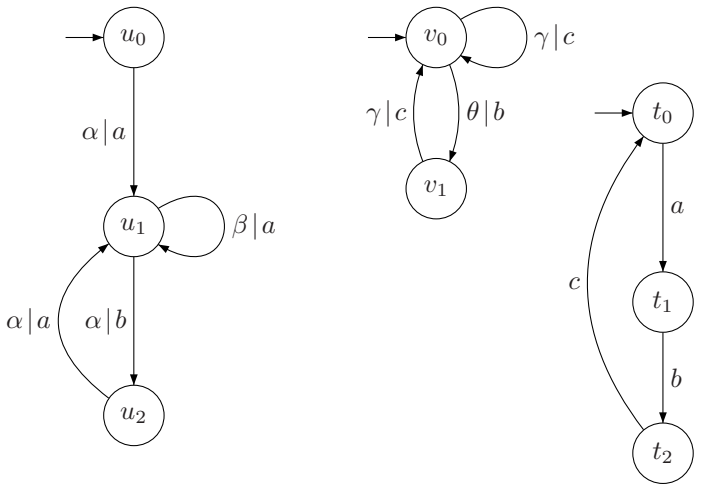


Figure 2.6: An example composition that admits more than one solution.

Not all the above mentioned relations are actually useful.

The reason is that any community state, except (u_0, v_1) and (u_2, v_1) can simulate all three target states. Furthermore, the two remaining states can simulate the target states t_0 and

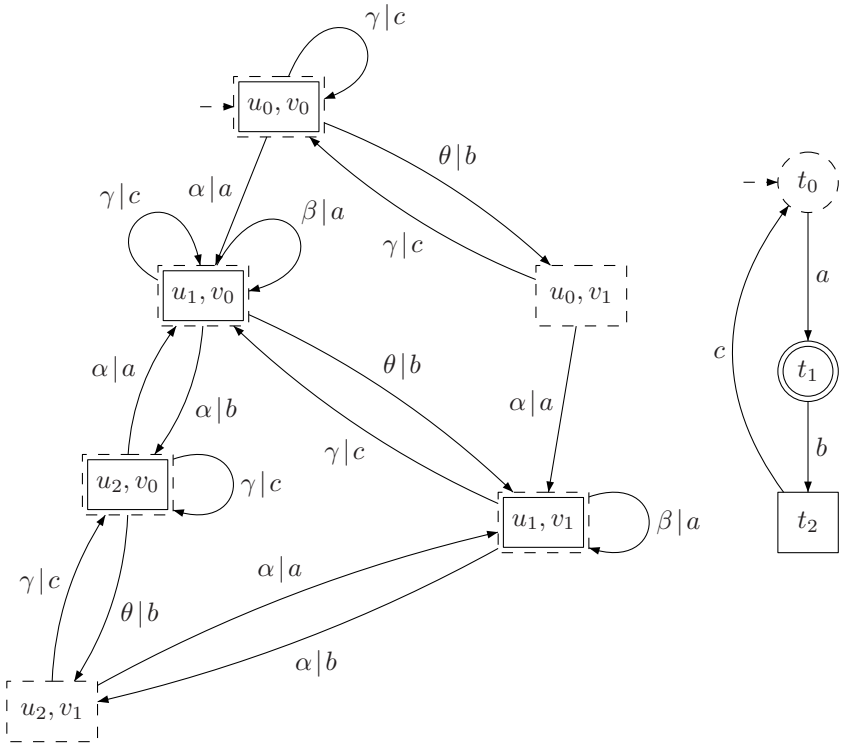


Figure 2.7: Example controllability relation corresponding to the example shown in Figure 2.6

t_2. Therefore the largest controllability which is the union of all controllability includes all community states and such a relation has 16 pairs out of the 18 possible pairs one could get from the Cartesian product of the community states with the target states. Such a relation is not useful however because many of the states are unreachable as can be seen in Figure 2.8 which shows the controllability graph. In the graph rectangular nodes are unreachable from the initial state $(u_0, v_0), t_0$ whereas the oval nodes are reachable. This example clearly shows that computing the **largest** controllability relation is not necessary and sometimes contains redundant information. Also, even though there a finite number of controllability relations, the number of orchestrators is infinite. This can be readily seen from the controllability graph shown in Figure 2.8. Whenever the system is in state $(u_1, v_0), t_1$ it can make two different choices in response of a "b" transition by the target: either sends an α message and move to state $(u_2, v_0), t_2$ or send an β message and move to state $(u_1, v_1), t_2$. Recall that the orchestrator values $\Omega(s, \tau, \alpha)$ are extracted from the controllability graph. Since there are an infinite number of histories ending in $((u_1, v_0), t_1)$ and for each of them there are two choices then the number of possible orchestrators is also infinite.

This informal discussion that relates the existence of an controllability to the existence of an orchestrator will be shown formally below.

To prove theorem 2.3.4 we need the following lemma.

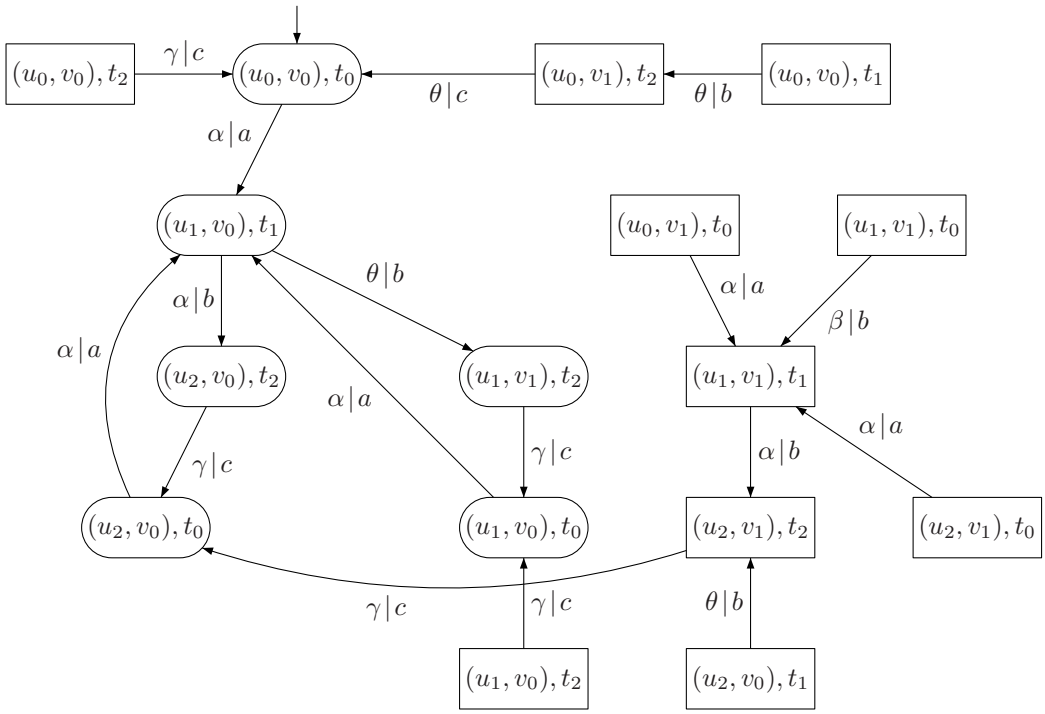


Figure 2.8: controllability graph corresponding to the controllability relation shown in Figure 2.7

Lemma 2.3.3. *Let R be an controllability relation between a target $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, \delta_t \rangle$ and a community $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$. Then there exists an orchestrator with perfect information Ω such that for all $(t, s) \in R$, $t \in S_t$,$s \in S$, and for all $a \in \Sigma = \Sigma_u \cup \Sigma_t$ we have $\delta_t(t, a) \neq \emptyset \Leftrightarrow \delta_\Omega(s, a) \neq \emptyset$. Furthermore for all $t' \in \delta_t(t, a)$ and $s' \in \delta_\Omega(s, a)$ we have $(t', s') \in R$.*

Proof. Let $(t, s) \in R$ where $s \in \Delta_\Omega(s^0, \tau)$ for some trace τ.

($\Rightarrow$) Suppose that $\delta_t(t, a) \neq \emptyset$. This means that $\exists t'.t \xrightarrow{a} t'$ and by the properties of R we have

- Either $\delta_u(s, a) \neq \emptyset$ and for all $s' \in \delta_u(s, a)$ we have $(t', s') \in R$. Therefore $\delta_\Omega(s, a) \supseteq \delta_u(s, a) \neq \emptyset$ and for all $t' \in \delta_t(t, a)$ and $s' \in \delta_\Omega(s, a)$ we have $(t', s') \in R$.

- Or $\exists \alpha \in E(s, t).\delta_u(\alpha \mid a) \neq \emptyset$ and for all $s' \in \delta_u(s, \alpha \,|\, a)$ we have $(t', s') \in R$. Recall that

$$\delta_\Omega(s, a) = \delta_u(s, a) \bigcup_\beta \delta_u(s, \beta \mid a) \odot \Omega(s, \tau, \beta)$$

  Thus if we choose $\Omega(s, \tau, \alpha) = 1$ then $\delta_\Omega(s, a) \supseteq \delta_u(s, a \mid \alpha) \odot \Omega(s, \tau, \alpha) \neq \emptyset$ and for all $s' \in \delta_\Omega(s, a)$ we have $(s', t') \in R$.

In both cases above $\delta_t(t,a) \neq \emptyset \Rightarrow \delta_\Omega(s,a) \neq \emptyset$ and for all $t' \in \delta_t(t,a)$ and $s' \in \delta_\Omega(s,a)$ we have $(t',s') \in R$.

($\Leftarrow$) Suppose that $\delta_\Omega(s,a) \neq \emptyset$. Let $s' \in \delta_\Omega(s,a)$ then by the definition of $\delta_\Omega(s,a)$ we have

- Either $s' \in \delta_u(s,a)$ then by the properties of R there exists $t'.t \xrightarrow{a} t'$ which means that $\delta_t(t,a) \neq \emptyset$. Furthermore, from R we have $(t',s') \in R$
- Or $s' \in \delta_u(s, \alpha \mid a)$ and $\Omega(s,\tau,\alpha) = 1$ for some $\alpha \in Com$. But according to our procedure all the messages for which the orchestrator is enabled are in $E(s,t)$ thus $\alpha \in E(s,t)$. Therefore from the definition of R we know that $\exists t'.t \xrightarrow{a} t'$ which means that $\delta_t(t,a) \neq \emptyset$. Furthermore $(t',s') \in R$.

Therefore we have shown that if $(t,s) \in R$ then for all $a \in \Sigma$ we have

$$\delta_t(t,a) \neq \emptyset \Leftrightarrow \delta_\Omega(s,a) \neq \emptyset$$

Furthermore for all $t' \in \delta_t(t,a)$ and $s' \in \delta_\Omega(s,a)$ we have $(t',s') \in R$. □

The above lemma will help us prove the following important theorem that characterizes the existence of an orchestrator in terms of the existence of controllability relation between the community and the target. Apart from the proposed model the following theorem is the first contribution of this thesis to the behavior composition problem.

Theorem 2.3.4. *Given a target service $\mathcal{S}_t = \langle S_t, \Sigma_t, s_t^0, \delta_t \rangle$ and a community of n available services $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$, then an orchestrator with perfect information Ω exists such that $\mathcal{S}_\Omega$ is a behavior composition of $\mathcal{S}_t$ iff $\mathcal{S}$ is controllable with respect to $\mathcal{S}_t$.*

Proof. ($\Rightarrow$) Suppose that $\mathcal{S}$ is controllable with respect to $\mathcal{S}_t$ then a controllability R relation exists between S_t and S. We need to prove that there exists an orchestrator Ω such that for all traces $\tau \in \Sigma^*$ and for all $t \in \Delta_t(t^0,\tau)$, $s \in \Delta_\Omega(s^0,\tau)$ and for all $a \in \Sigma$ we have $\delta_t(t,a) \neq \emptyset \Leftrightarrow \delta_\Omega(s,a) \neq \emptyset$. First, by lemma 2.3.3 an orchestrator Ω exists. Furthermore, Ω has the properties given in lemma 2.3.3. We use induction on the length of the trace τ to show that the orchestrated community $\mathcal{S}_\Omega$ is a composition of $\mathcal{S}_t$.

Base case. Consider the empty trace ϵ. Then $t \in \Delta_t(t^0,\epsilon)$ and $s \in \Delta_\Omega(s^0,\epsilon)$. Since $(t^0,s^0) \in R$ therefore by lemma 2.3.3, for all $a \in \Sigma$, $\delta_t(t^0,a) \neq \emptyset \Leftrightarrow \delta_\Omega(s^0,a) \neq \emptyset$. Also by lemma 2.3.3, for all $t \in \delta_t(t^0,a)$ and $s \in \delta_\Omega(s^0,a)$ we have $(t,s) \in R$. Therefore the base case is true.

Hypothesis: Assume that the above two properties are true for traces τ of length $l-1$. This implies that for arbitrary $t^{l-1} \in \Delta_t(t^0,\tau)$, $s^{l-1} \in \Delta_\Omega(s^0,\tau)$, for all $a \in \Sigma$ we have $\delta_t(t^{l-1},a) \neq \emptyset \Leftrightarrow \delta_\Omega(s^{l-1},a) \neq \emptyset$. Moreover, for all $a \in \Sigma$, $t^l \in \delta_t(t^{l-1},a)$, $s^l \in \delta_\Omega(s^{l-1},a)$ we have $(t^l,s^l) \in R$.

Induction step: Consider a trace τa of length l with $t^l \in \Delta_t(t^0, \tau a)$ and $s^l \in \Delta_\Omega(s^0, \tau a)$. From the defintion of the extended function Δ we know that $t^l \in \delta_t(t^{l-1}, a)$ and $s^l \in \delta_\Omega(s^{l-1}, a)$ for some $t^{l-1} \in \Delta_t(t^0, \tau)$ and $s^{l-1} \in \Delta_\Omega(s^0, \tau)$. Since τ is of length $l-1$ then by the induction hypothesis $(t^l, s^l) \in R$. It follows by lemma 2.3.3 that for all $a \in \Sigma$ we have $\delta_t(t^l, a) \neq \emptyset \Leftrightarrow \delta_\Omega(s^l, a) \neq \emptyset$.

($\Leftarrow$). Assume that a composition with orchestrator Ω exists. We need to show that an controllability relation between the target and the community exits.
Let $\tau \in \Sigma^*$ be an arbitrary trace of the target. Define the relation $R \subseteq S_t \times S$ relating the states of the target to the states of the community by

$$R = \{(t, s) \in S_t \times S \mid \exists \tau \in \Sigma^*, s \in \Delta_\Omega(s^0, \tau) \wedge t \in \Delta_t(t^0, \tau)\}$$

Note that since the target is deterministic then $\Delta_t(t^0, \tau)$ contains a single state t but we retain the notation for generality. Next we show that R is a controllability relation.

Let $(t, s) \in R$ and $t \xrightarrow{a} t'$ which means $\delta_t(t, a) \neq \emptyset$. Since a composition exists then $\delta_\Omega(s, a) \neq \emptyset$. Recall that

$$\delta_\Omega(s, a) = \delta_u(s, a) \bigcup_{\alpha \in Com} \delta_u(s, \alpha \mid a) \odot \Omega$$

Therefore the fact that $\delta_\Omega(s, a) \neq \emptyset$ implies that

- Either $\delta_u(s, a) \neq \emptyset$. Furthermore, from the definition of Δ we get that for all $s' \in \delta_u(s, a) \subseteq \delta_\Omega(s, a)$ then $s' \in \Delta_\Omega(s^0, \tau a)$ and $t' \in \Delta_t(t^0, \tau a)$, therefore $(t', s') \in R$.

- Or $\exists \alpha \in Com.\ \delta_u(s, \alpha \,|\, a) \neq \emptyset \wedge \Omega(s, \tau, \alpha) = 1$. This means that $\exists \alpha, s'.s \xrightarrow{\alpha|a} s'$ and since $s' \in \delta_u(s, \alpha \,|\, a) \subseteq \delta_\Omega(s, a)$ then $s' \in \Delta_\Omega(s^0, \tau a)$ and therefore $(t', s') \in R$.

  Furthermore, for all $b \in \Sigma - \{a\}$ consider an arbitrary $s'' \in \delta_u(s, \alpha \,|\, b)$. Since $\Omega(s, \tau, \alpha) = 1$ then $\delta_\Omega(s, b) \neq \emptyset$. But a composition exists then $\delta_t(t, b) \neq \emptyset$ and $\exists t''.t \xrightarrow{b} t''$. Also $s'' \in \Delta_\Omega(s^0, \tau b)$ and $t'' \in \Delta_t(t^0, \tau b)$ therefore $(t'', s'') \in R$. This means that α is such that for all $s \xrightarrow{\alpha|a} s' \Rightarrow \exists t'.t \xrightarrow{a} t' \wedge (t', s') \in R$. Therefore $\alpha \in E(s, t)$.

Conversely suppose that $\delta_u(s, a) \neq \emptyset$. Then $\delta_\Omega(s, a) \supseteq \delta_u(s, a) \neq \emptyset$ and since a composition exists then $\delta_t(t, a) \neq \emptyset$ and thus $\exists t'.t \xrightarrow{a} t'$. Let $s' \in \delta_u(s, a) \subseteq \delta_\Omega(s, a)$ then from the definition of Δ we have $s' \in \Delta_t(s^0, \tau a)$ and $t' \in \Delta_\Omega(t^0, \tau a)$ and therefore $(t', s') \in R$.

Combining the above partial results we get that R is a controllability relation.

□

2.3.3 Relation to control theory

So far we have formulated the behavior composition problem as done in [GPS13]. An alternative definition, similar to the classical control theory, will be given in this section and the relationship between the two is explored.

Theorem 2.3.5. *Let $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ be a community of n services and $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, \delta_t \rangle$ be a target service. Also let Ω be an orchestrator and $\mathcal{S}_\Omega = \langle S, \Sigma, s^0, \delta_\Omega \rangle$ be the orchestrated community and denote bisimulation equivalence by $\equiv$. Then $\mathcal{S}_\Omega$ is a behavior composition of $\mathcal{S}_t$ iff $\mathcal{S}_\Omega \equiv \mathcal{S}_t$*

Proof. Suppose that $\mathcal{S}_\Omega$ is a behavior composition of $\mathcal{S}_t$. Let $\tau \in \Sigma^*$ denote an arbitrary sequence of actions. Define the relation R between $\mathcal{S}_\Omega$ and $\mathcal{S}_t$ as

$$R = \{(s,t) \in S \times S_t \mid \exists \tau \in \Sigma^*, s \in \Delta_\Omega(s^0, \tau) \wedge t \in \Delta_t(t^0, \tau)\}$$

We show that R is a bisimulation relation. Consider an arbitrary transition $t \xrightarrow{a} t'$ then $\delta_t(t,a) \neq \emptyset$ and by definition 2.3.3 this implies $\delta_\Omega(s,a) \neq \emptyset$. Let $s' \in \delta_\Omega(s,a)$ then by the definition of Δ we have that $t' \in \Delta_t(t^0, \tau a)$ and $s' \in \Delta_\Omega(s^0, \tau a)$ and thus $(s',t') \in R$. We have show that for all $t \xrightarrow{a} t'$ $\exists s'.s \xrightarrow{a} s'$ and $(s',t') \in R$. The reverse can be shown by symmetry and therefore R is a bisimulation. Finally, since $s^0 \in \Delta_\Omega(s^0, \epsilon)$ and $t^0 \in \Delta_\Omega(t^0, \epsilon)$ then $(s^0, t^0) \in R$ and therefore $\mathcal{S}_\Omega \equiv \mathcal{S}_t$.

Conversely, assume that $\mathcal{S}_\Omega \equiv \mathcal{S}_t$ and let R be the bisimulation relation. We need to show that for all $s \in \Delta_\Omega(s^0, \tau)$, $t \in \Delta_t(t^0, \tau)$ we have $\delta_\Omega(s,a) \neq \emptyset \Leftrightarrow \delta_t(t,a) \neq \emptyset$. This is done by induction on the length of τ.

Base case: $\tau = \epsilon$. Since $(s^0, t^0) \in R$ then for all $a \in \Sigma$, if $t^0 \xrightarrow{a} t^1$ then $\exists s^1.s^0 \xrightarrow{a} s^1$, thus $\delta_t(t^0, a) \neq \emptyset \Rightarrow \delta_\Omega(s^0, a) \neq \emptyset$. Furthermore, because R is a bismulation then $(s^1, t^1) \in R$. Similarly, $\delta(s^0, a) \neq \emptyset \Rightarrow \delta(t^0, a) \neq \emptyset$.

Hypothesis: Assume that the above is true for all traces,τ, of size l i.e.: $s \in \Delta_\Omega(s^0, \tau) \wedge t \in \Delta_t(t^0, \tau) \Rightarrow \delta_\Omega(s,a) \neq \emptyset \Leftrightarrow \delta_t(t,a) \neq \emptyset$ and for all $s' \in \delta_\Omega(s,a)$, $t' \in \delta_t(t,a)$ we have $(s',t') \in R$.

Induction step: Consider a trace, τa of size $l+1$. Let $s \in \Delta_\Omega(s^0, \tau a)$ and $t \in \Delta_t(t^0, \tau a)$. From the definition of Δ, we know that $\exists s' \in \Delta_\Omega(s^0, \tau), t' \in \Delta_t(t^0, \tau)$ such that $t \in \delta_t(t', a)$ and $s \in \delta_\Omega(s', a)$. By the induction hypothesis $(s,t) \in R$ and it follows from the definition of a bisimulation that $\delta_\Omega(s,a) \neq \emptyset \Leftrightarrow \delta_t(t,a) \neq \emptyset$ □

One can define other equivalences between the orchestrated community and the target service. For example, it is not hard to show that if in definition 2.3.3 one replaces "$\Leftrightarrow$" by a simple implication: $\delta(t,a) \neq \emptyset \Rightarrow \delta(s,a) \neq \emptyset$ then the orchestrated community *simulates* the target, i.e. $\mathcal{S}_\Omega \sqsubseteq \mathcal{S}_t$ where $\sqsubseteq$ is the simulation relation.

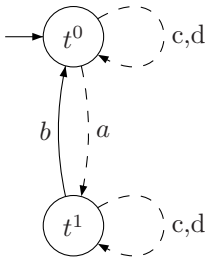


Figure 2.9: Before an a transition no b transition is permitted. After an a transition every state must be able to make a b transition, until then no a transition is possible. After a b transition a transitions are allowed again while b is not permitted

2.3.4 Modal specifications

So far the aim of the behavior composition was to satisfy a *single* target behavior. We can extend this aim by satisfying a *set* of target behaviors by using *modal specifications*. Modal specifications have been introduced to model control problem objectives in [FP07]. The main idea behind modal specifications is that some transitions while allowed are not strictly necessary (called May transitions) while others are strictly necessary (called Must transitions). A simple example of a modals specification is shown in Figure 2.9 below.

Definition 2.3.6 (Modal specification)**.** *A modal specification is a tuple* $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, May, Must \rangle$ *where*

- S_t *is a set of finite states.*
- t^0 *is the initial state.*
- Σ_t *is a finite set of actions.*
- $May \subseteq S_t \times \Sigma \times S_t$ *is a deterministic transition relation of allowed transitions.*
- $Must \subseteq May$ *is a deterministic transition relation of necessary transitions.*

A modal specification is said to be deterministic iff the May transition is deterministic. Having defined the specification next we give the composition problem.

Definition 2.3.7 (Behavior composition with modal specification)**.** *Let* $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ *be a community of available services and* $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, May, Must \rangle$ *be a modal specification. Let* Ω *be an orchestrator and denote by* $S_\Omega = \langle S, \Sigma_u, s^0, \delta_\Omega \rangle$ *the orchestrated community. We say that* S_Ω *is a behavior composition of modal specification* $\mathcal{S}_t$ *iff there exists a relation* $\rho \subseteq S \times S_t$ *such that for all* $(s,t) \in \rho$ *and all* $a \in \Sigma = \Sigma_u \cup \Sigma_t$ *we have:*

- $(t, a, t') \in Must \Rightarrow \delta_\Omega(s,a) \neq \emptyset \wedge \forall s' \in \delta_\Omega(s,a), (s', t') \in \rho$

- $\delta_\Omega(s,a) \neq \emptyset \Rightarrow \exists t'.(t,a,t') \in May \wedge \forall s' \in \delta_\Omega(s,a), (s',t') \in \rho$

The characterization of the existence of a composition in the case of modal specification is similar to previous cases and depends on the concept of controllability.

Definition 2.3.8 (Controllability with respect to modal specification)**.** *Let $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ be a community of available services and $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, May, Must \rangle$ be a modal specification. We say that $\mathcal{S}$ is controllable with respect to $\mathcal{S}_t$ iff there exists a relation $R \subseteq S \times S_t$ such that $(s^0, t^0) \in R$ and for all $(s,t) \in R$ we have*

- $s \xrightarrow{a} s' \Rightarrow \exists t'.(t,a,t') \in May \wedge (s',t') \in R$
- $\exists E(s,t) \subseteq Com$ *such that for all* $\alpha \in E, b \in \Sigma$ *we have* $s \xrightarrow{\alpha|b} s' \Rightarrow \exists t'.(t,b,t') \in May \wedge (s',t') \in R$
- $(t,a,t') \in Must \Rightarrow$
  - *Either* $\delta_u(s,a) \neq \emptyset$ *and for all* $s' \in \delta_u(s,a)$ *we have* $(s',t') \in R$
  - *Or* $\exists \alpha \in E(s,t)$ *such that* $\delta_u(s, \alpha \,|\, a) \neq \emptyset$.

The following theorem is a generalization of previous results.

Theorem 2.3.6. *Let $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ be a community of available services and $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, May, Must \rangle$ be a modal specification. An orchestrator Ω exists such that the orchestrated community $S_\Omega = \langle S, \Sigma_u, s^0, \delta_\Omega \rangle$ is a behavior composition of modal specification $\mathcal{S}_t$ iff $\mathcal{S}$ is controllable with respect to $\mathcal{S}_t$.*

Proof. Assume that $\mathcal{S}$ is controllable with respect to $\mathcal{S}_t$ and let R be the controllability relation. We construct Ω such that $\mathcal{S}_\Omega$ is composition with respect to $\mathcal{S}_t$. To do so, we show how to construct an orchestrator Ω such that the following relation

$$\rho = \{(s,t) \in R \mid \exists \tau \in \Sigma^*,\ s \in \Delta_\Omega(s^0, \tau), t \in \Delta_t(t^0, \tau)\}$$

has the properties given in definition 2.3.7. Choose an arbitrary $(s,t) \in \rho$ with $s \in \Delta_\Omega(s^0, \tau)$,$t \in \Delta_t(t^0, \tau)$ for some $\tau \in \Sigma^*$. We have:

- Suppose that $(t,a,t') \in Must$. Since $(s,t) \in R$ then by the properties of R in definition 2.3.8 we have:
  - Either $\delta_u(s,a) \neq \emptyset$ and for all $s' \in \delta_u(s,a)$ we have $(s',t') \in R$. In this case $\delta_\Omega(s,a) \supseteq \delta_u(s,a) \neq \emptyset$. Also, from the definition of Δ we have $s' \in \Delta_\Omega(s^0, \tau a)$ and $t' \in \Delta_t(t^0, \tau a)$ therefore $(s',t') \in \rho$ for all $s' \in \delta_\Omega(s,a)$.
  - Or $\exists \alpha \in E(s,t) \subseteq Com, s' \in S. s \xrightarrow{\alpha|a} s' \wedge (s',t') \in R$. In this case choose $\Omega(s,\tau,\alpha) = 1$ then $\delta_\Omega(s,a) \supseteq \bigcup_{\alpha \in Com} \delta_u(s, \alpha \mid a) \odot \Omega(s,\tau,\alpha) \neq \emptyset$ and for all $s' \in \delta_\Omega(s,a), (s',t') \in R$. Furthermore, $s' \in \Delta_\Omega(s^0, \tau a)$ and $t' \in \Delta_t(t^0, \tau a)$ thus $(t',s') \in \rho$.

- Suppose that $s' \in \delta_\Omega(s,a)$ then
  - Either $s' \in \delta_u(s,a)$ then by the property of R, $\exists t'.(t,a,t') \in May$ and $(s',t') \in R$. Again $s' \in \Delta_\Omega(s^0, \tau a)$ and $t' \in \Delta_t(t^0, \tau a)$ thus $(t',s') \in \rho$.
  - Or $s' \in \delta_u(s, \beta \mid a)$ and $\Omega(s,t,\beta) = 1$ for some $\beta \in Com$. Now by the above construction $\Omega(s,t,\beta)$ is set to 1 only if $\beta \in E(s,t)$. Then by the definition of $E(s,t)$ we have $\exists t'.(t,a,t') \in May$ and $(s',t') \in R$. Also $s' \in \Delta_\Omega(s^0, \tau a)$ and $t' \in \Delta_t(t^0, \tau a)$ thus $(t',s') \in \rho$.

  Therefore $\mathcal{S}_\Omega$ is a behavior composition of $\mathcal{S}_t$.
  ($\Leftarrow$) Suppose that for some Ω, $\mathcal{S}_\Omega$ is a behavior composition of modal specification $\mathcal{S}_t$. Let ρ be the composition relation. Next we show that ρ is a controllability relation between $\mathcal{S}$ and $\mathcal{S}_t$. Let $(s,t) \in \rho$ and $a \in \Sigma$ an arbitrary action
  - Suppose that $(t,a,t') \in Must$. Because ρ is a composition relation it follows from the first item in definition 2.3.7 that $\delta_\Omega(s,a) \neq \emptyset$ and for all $s' \in \delta_\Omega(s,a)$ we have $(s',t') \in \rho$. From the definition of $\delta_\Omega(s,a)$ we have two cases
    - Either $\delta_u(s,a) \neq \emptyset$ and for all $s' \in \delta_u(s,a)$ we have $(s',t') \in \rho$.
    - Or for some α, $\delta_u(s, \alpha \mid a) \neq \emptyset$ and for all $s' \in \delta_u(s, \alpha \mid a)$ we have $(s',t') \in \rho$. Furthermore $\Omega(s,t,a) = 1$. We still need to show that $\alpha \in E(s,t)$. Let $b \in \Sigma$ such that $\delta_u(s, \alpha \,|\, b) \neq \emptyset$. Since $\Omega(s,t,\alpha) = 1$ then $\delta_\Omega(s,b) \neq \emptyset$. But ρ is a composition relation then by the second item in definition 2.3.7, $\exists t'.(t,b,t') \in May$ and for all $s' \in \delta_u(s, \alpha \,|\, b)$ we have $(s',t') \in \rho$.
  - Suppose that $s \xrightarrow{a} s'$, i.e. $s' \in \delta_u(s,a)$, then by construction $\delta_\Omega(s,a) \neq \emptyset$. Because ρ is a composition relation it follows from the second item in definition 2.3.7 $\exists t'.(t,a,t') \in May$ with $(s',t') \in \rho$.

□

The above result opens up many possibilities. First it is easy to seen that if $Must = May$ then requiring the orchestrated community $\mathcal{S}_\Omega$ to satisfy a modal specification $\mathcal{S}_t$ is the same as requiring the two to be bisimilar. In the same manner one can show that the simulation relation is also a special case of modal specification. This means that theorem 2.3.6 is a general result that is applicable to many different settings.

The importance of modal specification goes beyond being a generalization. It is a natural way to express a *set of behavior goals*. An important property of modal specification is their underlying logic they correspond to. In [FP07] it was shown that modal specification correspond to the *conjunctive mu-calculus*. This logic is a syntactic fragment of the modal mu-calculus of [Koz83] limited to the operators: $p, \neg, \wedge, [\,], <>$ and greatest fix-point.

We don't give the semantic of the logic in this thesis since we only use modal specifications. However, the logical framework highlight the useful property that one can combine multiple modal specifications with an *and* operator. For example given k modal specifications $\mathcal{S}_1 \ldots \mathcal{S}_k$

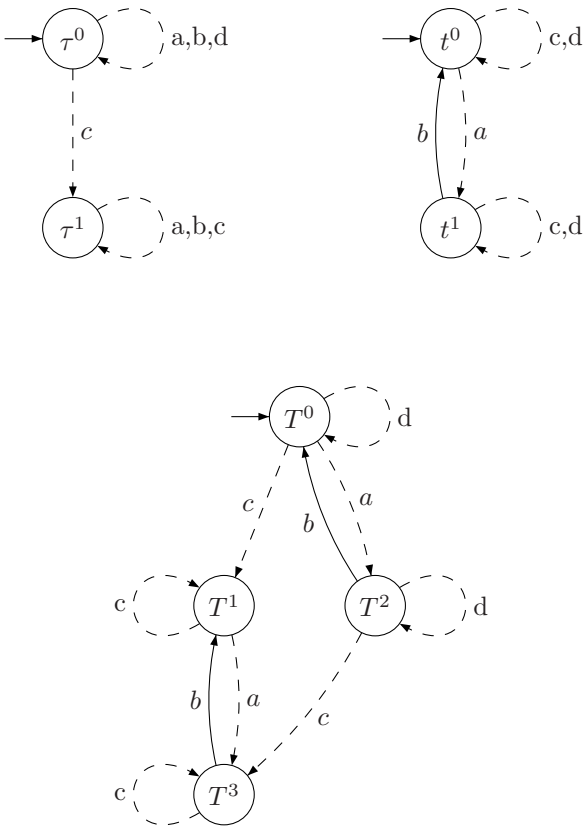


Figure 2.10: Two modal specifications, τ top left and t top right are "and" combined to produce the modal specification in the bottom T.

one merges the starting state of all $\mathcal{S}_i$ and performs a determinization procedure on the May transition to obtain the resulting specification $\mathcal{S}_t$ which is equivalent to the *and* of all specifications. Obviously the determinization procedure brings an exponential blowup. However, this also means that one can build a complex specification with a set of very concise combination of specifications. An example of the above procedure is shown in Figure 2.10. Two specifications τ to the top left of the figure, t to the top right of the figure are combined to produce the T specification in the bottom of the figure. Specification τ says that after a c is executed no d is possible. Specification t says that no b can be executed before an a is executed. Also once an a is executed a b must be executed before any other a. It is easy to verify that the combination of these specifications produces the T specification in the bottom of the figure.

2.4 Orchestrator with partial information

In this section we handle the case when some of the actions of the services are not observable. This could be the case when the services and the orchestrator are from different providers then the orchestrator does not necessarily have access to the services except through the communication action.

The orchestrator can send messages, to the community of services, from a set Com and "remembers" only the messages it has already sent. In other words, the orchestrator is assumed to observe only its own transitions. We explain further with an example. In Figure 2.11 below we show the community S, an orchestrator Ω, and the effect of the orchestrator on the community (i.e. *orchestrated community*). Initially the orchestrator is in state Ω_0, meaning it has not sent any message yet which we denote by the special message ϵ. The community can be in any of the states $\{s_0, s_1, s_3, s_4\}$. Then the observable of those states is the same and is equal to ϵ because all of those states can be reached from the initial state by receiving the empty message ϵ. Similarly, By sending a message α the controller can cause the state s_0 to transition to state s_2, and state s_4 to transition to state s_6. All other states are unaffected. This means that if the orchestrator sends α then the system can be in states $\{s_2, s_6\}$. The observable of these two states is α. In reality the observable string is a property of the trace in addition to the state. In the example in Figure 2.11 the two coincide because there is a single trace associated with each state. Formally,

Definition 2.4.1 (Message trace)**.** *The message trace is a function $\sigma : S \times \Sigma^* \longrightarrow Com^*$ that associates with a state $s \in S$ reached via a sequence of actions $\tau \in \Sigma^*$ the sequence of messages $\sigma(s, \tau) \in Com^*$.*

For example $\sigma(s^0, \epsilon) = \sigma(s^1, a) = \sigma(s^4, aa) = \epsilon$, the empty string, whereas the message trace to reach state s^7 is $\sigma(s^7, aabc) = \beta\alpha$. Since the orchestrator is deterministic then the state of an orchestrator can be represented by (not necessarily unique) message trace. For example Ω_0 is represented by the empty string, and Ω_1 is represented by α. In what follows we use the function $\Omega(x, y)$ where $x \in Com^*$ and $y \in Com$ to mean an orchestrator transition from state represented by string x on y. In Figure 2.11, $\Omega(\epsilon, \alpha) = 1$ and all other values are 0. Let $\delta_u(s, a)$ represent the transition of the community on action a then the evolution of the community is described as:

$$\Delta_\Omega(s_0, \tau a) = \bigcup_{s \in \Delta(s^0, \tau)} \delta_\Omega(s, a)$$

where

$$\delta_\Omega(s, a) = \delta_u(s, a) \bigcup_{\alpha \in Com} \delta_u(s, \alpha \,|\, a) \odot \Omega\left(\sigma(s, \tau), \alpha\right)$$

and

$$\delta_u(s, \alpha \,|\, a) \odot \Omega(\sigma, \alpha) = \begin{cases} \emptyset & \text{If } \Omega(\sigma, \alpha) = 0 \\ \delta_u(s, \alpha \,|\, a) & \text{If } \Omega(\sigma, \alpha) = 1 \end{cases}$$

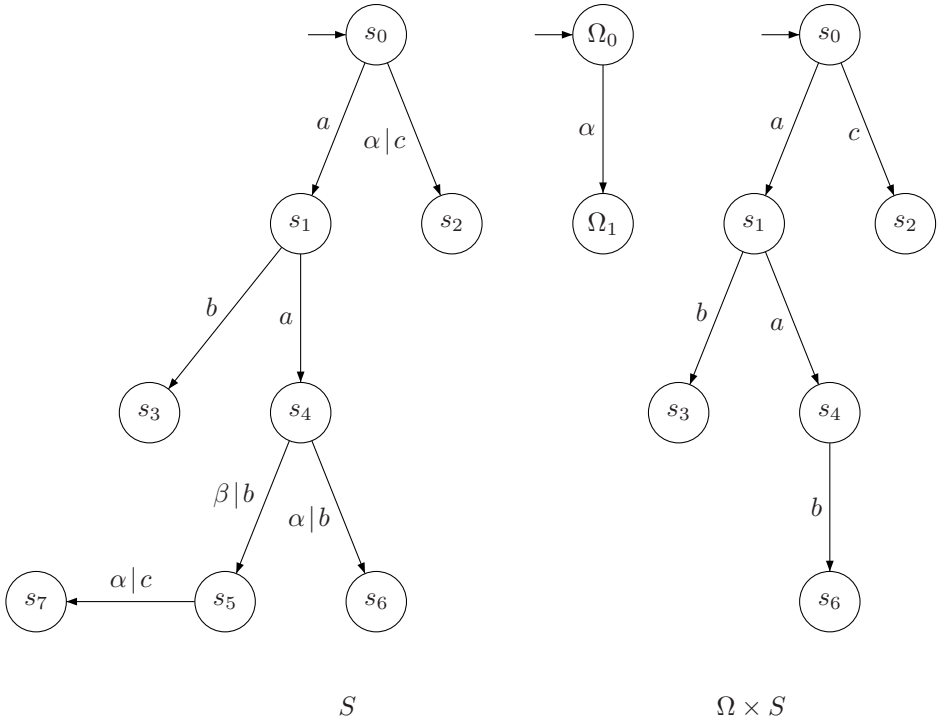


Figure 2.11: Interaction of an Orchestrator with partial observation with a service

This is the same formulation as before except the dependence of the orchestrator on the message trace function σ. Note that the value of σ is in Com^*.

Definition 2.4.2 (Behavior composition with partial information)**.** *Let $\mathcal{S}_t$ be a target service and $\mathcal{S}$ be a community of services. Let Ω be an orchestrator with partial information and denote by $\mathcal{S}_\Omega$ the orchestrated community. We say that $\mathcal{S}_\Omega$ is a behavior composition of $\mathcal{S}_t$ with partial information iff for all traces $\tau \in \Sigma^*$ and all $t \in \Delta_t(t^0, \tau)$, $s \in \Delta_\Omega(s^0, \tau)$ we have :*

$$\forall a \in \Sigma, \quad \delta_t(t, a) \neq \emptyset \Leftrightarrow \delta_\Omega(s, a)$$

Again, with the exception of σ, this is the same definition as in definition 2.3.3. Similarly to the case of perfect information, the existence of an orchestrator is characterized by the existence of a set of relations instead of a single relation. Before giving the formal definition it helps to present an example. Figure 2.12 shows an example community service that realizes the target with the help of an orchestrator with partial observation. First note that service state s^4 has different "observable" values depending on the path followed. The interaction of the orchestrator with the community can be seen as a *synchronous* product of the two, written as $S \times \Omega$ where

S and Ω are the LTSs of the community and the orchestrator respectively. Such synchronous product is defined as: the community can make a "spontaneous" transition whilst the orchestrator is still, or the community can make a preconditioned transition in sync with the orchestrator if the orchestrator can enable that precondition. This synchronous product is shown in Figure 2.13 together with the target. It is clear from the target that the resulting service realizes the target.

To be able to characterize the problem as we have done before it is useful to look at it from the perspective of observability of states, i.e. combining states with the same observability together. Using Figure 2.12 as a guide, when the orchestrator is in state Ω_0, no action is performed, the community can be in states$\{s^0, s^1\}$ only. More precisely, the orchestrator cannot observe the uncontrollable transition $s^0 \xrightarrow{a} s^1$. Each of those states is "paired" with a target state to get the set of pairs $\Pi_0 = \{(s^0, t^0), (s^1, t^1)\}$ with the property that for every pair (s, t) if $s \xrightarrow{a} s'$ then $\exists t'.t \xrightarrow{a} t'$ and (s', t') belong to the same set Π_0. Now when the α action is enabled, i.e. the orchestrator sends message α, then the service can be in states $\{s^3, s^0, s^1, s^4, s^2\}$. This is because α makes s^1 transition to s^3 and s^0 to s^2 but once in s^3 or s^2 the service can make unobserved transitions from s^3 to $\{s^0, s^1\}$ and from s^2 to s^4. Each of those states is paired with the target state to get the set $\Pi_1 = \{(s^0, t^0), (s^1, t^1), (s^2, t^2), (s^3, t^3), (s^4, t^4)\}$. Subsequently any action α will not change anything in the set of states of the service. Finally, when in set Π_0 or Π_1 and a message γ is sent then the community could be in state $\{s^4\}$ (because both Π_0 and Π_1 contain the state s^1) which paired with t^4 we get a new set $\Pi_2 = \{(s^4, t^4)\}$. Let $Z = \{\Pi_0, \Pi_1, \Pi_2\}$ then Z has the following property: for all $\Pi_i \in Z$ and for all $(s, t) \in \Pi_i$ if $s \xrightarrow{a} s'$ then $\exists t'.t \xrightarrow{a} t'$ and $(s', t') \in \Pi_i$. Also for all $(s, t) \in \Pi_i$ if $t \xrightarrow{a} t'$ then either $\exists s'.s \xrightarrow{a} s'$ and $(s', t') \in \Pi_i$ or $\exists s', \alpha.s \xrightarrow{\alpha|a} s'$ and $(s', t') \in \Pi_j$ for some $\Pi_j \in Z$. One can see that the sets $\Pi_i \in Z$ are just the states of the orchestrator.

Another useful observation is the following. For each state s let θ_s be a regular expression, describing the (infinite) set of observable strings for that state. For example in Figure 2.12, $\theta_{s^0} = \theta_{s^1} = \alpha^*$, $\theta_{s^2} = \theta_{s^3} = \alpha^+$ and $\theta_{s^4} = \alpha^+ \cup \alpha^*\gamma$. The components of the regular expressions are then used to label the sets of states. Again in Figure 2.12 the individual components are $\{\epsilon, \alpha^+, \alpha^*\gamma\}$, we get $\Pi_{\alpha^*} = \{(s^0, t^0), (s^1, t^1)\}$, $\Pi_{\alpha^+} = \{(s^0, t^0), (s^1, t^1), (s^2, t^2), (s^3, t^3), (s^4, t^4)\}$, $\Pi_{\alpha^+ \cup \alpha^*\gamma} = \{(s^4, t^4)\}$. Note that these three sets are exactly Π_0, Π_1, Π_2 which constitutes the states of the orchestrators. We call these sets the *observation relations*. In the next section we give all the necessary formal definition to the concepts introduced in this section.

2.4.1 Observation relations

Before introducing the concept of *observation relation* we start with some needed definitions first.

Definition 2.4.3 (State regular expression)**.** *Given a community state s, then we associate a regular expression. θ_s, with s defined by the language it generates: $L(\theta_s) = \{x \in Com^* \mid \exists \tau \in \Sigma^*$ with $\sigma(s, \tau) = x\}$.*

That indeed a regular expression exists for every state, can be seen from the fact that all the strings x obtained as $\sigma(s, a^1 \dots a^l) = x$ are the language of the finite automaton obtained from

the community LTS whose only accepting state is s. By the equivalence of NFA's and regular expressions we know that a regular expression exits.

Definition 2.4.4 (Observable sets)**.** *Let Θ be the set of all regular expressions of the community. The observable sets, is a set of sets $Z = \{R_\theta \mid \theta \in \Theta\}$ where each R_θ is a set defined as:*

$$R_\theta = \{(t,s) \mid \forall x \in L(\theta) \exists \tau \in \Sigma^*.s \in \Delta(s^0,\tau) \wedge t \in \Delta(t^0,\tau) \\ \wedge \sigma(s,\tau) = x\}$$

Next we use the above to definitions to formalize the concept of *observation relations*. As we have seen in the previous section, each relation in such a set is a pair of states (s,t) one in the target and the other in the community. Given one *observation relation* R and a pair $(s,t) \in R$ then any uncontrollable transition $s \xrightarrow{a} s'$ from some community state is matched by a target transition $t \xrightarrow{a} t'$. Furthermore, $(s',t') \in R$. Now if the target state makes a transition $t \xrightarrow{a} t'$ it has to be matched by community transition either controllable $s \xrightarrow{\alpha|a} s'$ or uncontrollable $s \xrightarrow{a} s'$. There is an extra condition on the controllable transitions however. Since any enabling by the orchestrator of an α transition can potentially enable other transitions, it has to be done in such a way that all transitions end up in the same destination relation R'. Formally,

Definition 2.4.5 (Observation Relations)**.** *A set of relations $Z \subseteq 2^{S_t \times S}$ is called a set of observation relations iff : for every $R \in Z$ and for every $(t,s) \in R$ we have the following*

1. *If $t \xrightarrow{a} t'$ then*
   (a) *Either $\exists s'.s \xrightarrow{a} s' \wedge (t',s') \in R$*
   (b) *Or $\exists s', \alpha \in Com, R' \in Z$ with*

$$s \xrightarrow{\alpha|a} s' \wedge (t',s') \in R' \\ \wedge \\ \forall (u,v) \in R(v \xrightarrow{\alpha|b} v' \Rightarrow \exists u'.u \xrightarrow{b} u' \wedge (u',v') \in R')$$

2. *If $s \xrightarrow{a} s'$ then $\exists t'.t \xrightarrow{a} t' \wedge (t',s') \in R$*

This concept of a set of observation relations is similar to the concept of *regions* in Petri Nets [BD98]. The set of observation relations just defined will be used to define the concept of controllability under partial information.

Definition 2.4.6 (Controllability under partial onformation)**.** *A community of services $\mathcal{S}$ is said to be controllable under partial information with respect to a target service $\mathcal{S}_t$ iff there exists a set of observation relations Z between the states of $\mathcal{S}$ and $\mathcal{S}_t$ such that $\exists R_0 \in Z$ with $(s^0,t^0) \in R_0$.*

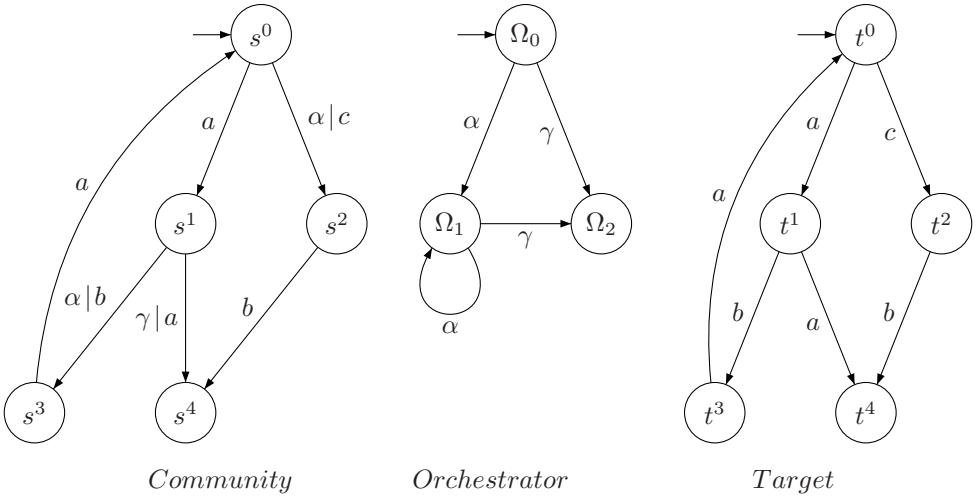


Figure 2.12: Orchestrator with partial observation realizing a target

The above definition allows us to characterize the existence of an orchestrator in a manner similar to the case of perfect information.

Theorem 2.4.1. *Let $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ be a community of services and $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, \delta_t \rangle$ be a target service. An orchestrator with partial information, Ω, exists such that $\mathcal{S}_\Omega$ is a composition of $\mathcal{S}_t$ iff $\mathcal{S}$ is controllable under partial information with respect to $\mathcal{S}_t$.*

Proof. ($\Rightarrow$). Suppose that $\mathcal{S}$ is controllable with respect to $\mathcal{S}_t$ then a set of relations Z having the properties in definition 2.4.6, exists. We show by induction on the length of an arbitrary trace τ that for all $t \in \Delta_t(t^0, \tau)$ and $s \in \Delta_\Omega(s^0, \tau)$ we have:

$$\delta_t(t, a) \neq \emptyset \Leftrightarrow \delta_\Omega(s, a) \neq \emptyset$$

Base case. Consider the empty trace ϵ then $t^0 \in \Delta_t(t^0, \epsilon)$ and $s^0 \in \Delta_\Omega(s^0, \epsilon)$. Also $(t^0, s^0) \in R$ for some $R \in Z$. Let $t = \delta_t(t^0, a)$ which means that $t^0 \xrightarrow{a} t$ then by the property of Z we have:

- Either $\exists s$ such that $s^0 \xrightarrow{a} s$ and $(t, s) \in R$.
- Or $\exists \alpha \in Com, s$ such that $s^0 \xrightarrow{\alpha|a} s$ and $(t, s) \in R'$ for some $R' \in Z$. Furthermore, for all $(u, v) \in R$ we have $v \xrightarrow{\alpha|b} v' \Rightarrow u \xrightarrow{b} u'$ with $(u', v') \in R'$. In this case we choose $\Omega(\sigma(s^0, \epsilon), \alpha) = 1$ then

$$\delta_\Omega(s^0, a) \supseteq \delta_u(s^0, \alpha \mid a) \odot \Omega(\sigma(s^0, \epsilon), \alpha) \neq \emptyset$$

We have shown that $\delta_t(t^0, a) \neq \emptyset \Rightarrow \delta(s^0, a) \neq \emptyset$.
Now we consider the opposite direction. Suppose that $s \in \delta_\Omega(s^0, a)$ then there are two cases:

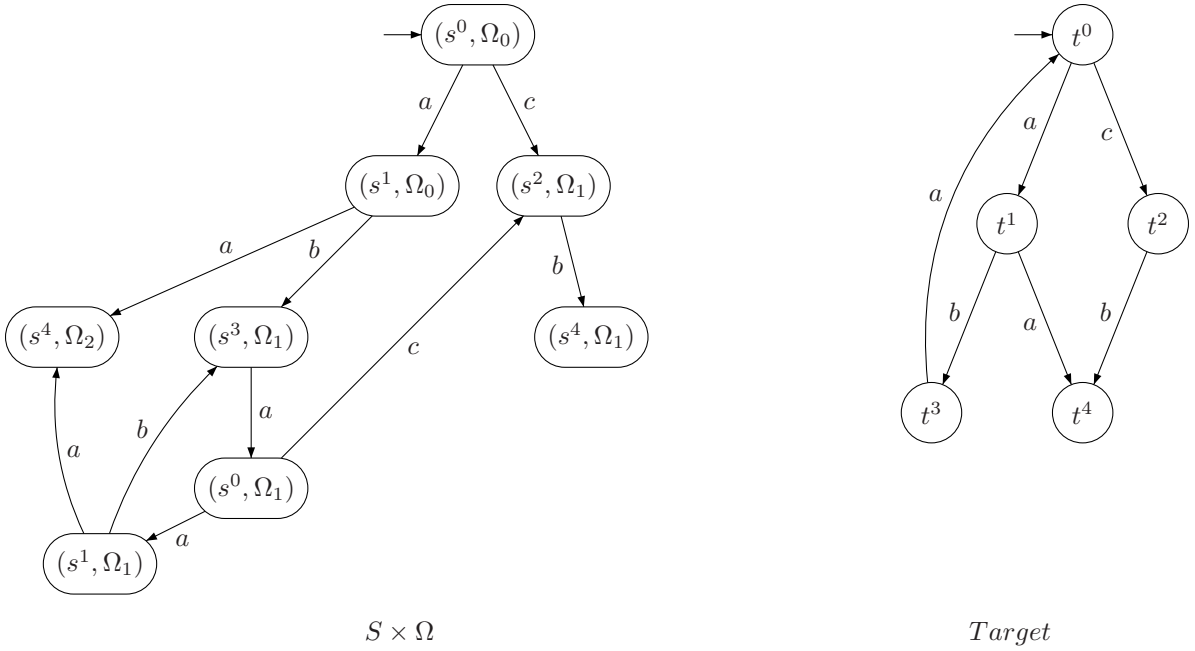


Figure 2.13: Synchronous product of service with orchestrator realizing the target

- Either $s^0 \xrightarrow{a} s$ then by Z, $\exists t$ with $t^0 \xrightarrow{a} t$ and $(t, s) \in R$.
- Or $s^0 \xrightarrow{\alpha|a} s$ and $\Omega(\sigma(s^0, \epsilon), \alpha) = 1$ for some α. Now $\Omega(s^0, \epsilon, \alpha) = 1$ means that α transitions were enabled and this is done only to match some target transition $t^0 \xrightarrow{b} t'$ by a community transition $s^0 \xrightarrow{\alpha|b} s'$. By the property of R we know that this is done in such that a way that for all $c \in \Sigma$ with $s^0 \xrightarrow{\alpha|c} v$ there exists u such that $t^0 \xrightarrow{c} u$ and $(u, v) \in R'$. In particular, $\exists t.t^0 \xrightarrow{a} t$ and $(t, s) \in R'$.

We have shown that $\delta_t(t^0, a) \neq \emptyset \Leftrightarrow \delta_\Omega(s^0, a) \neq \emptyset$. Moreover for all $s \in \delta_\Omega(s^0, a)$ and $t \in \delta_t(t^0, a)$ we have $(t, s) \in R$ for some $R \in Z$.

Induction hypothesis. Assume that the above properties are true for all traces, τ, of length $l - 1$. This means that for all $s \in \Delta_\Omega(s^0, \tau)$ and $t = \Delta_t(t^0, \tau)$ we have $\delta_t(t, a) \neq \emptyset \Leftrightarrow \delta_\Omega(s, a) \neq \emptyset$ and furthermore $(t, s) \in R$ for some $R \in Z$.

Induction step. Consider a trace, τb, of length l. Let $t^l \in \Delta_t(t^0, \tau b)$ and $s^l \in \Delta_\Omega(s^0, \tau b)$. Suppose that $t^l \xrightarrow{a} t^{l+1}$. From the definition of Δ we know that $t^l \in \delta_t(t, b)$ and $s^l \in \delta_\Omega(s, b)$ for some $s \in \Delta_\Omega(s^0, \tau)$ and $t \in \Delta_t(t^0, \tau)$. Since τ is of length $l - 1$ then by the induction hypothesis $(t^l, s^l) \in R$ and it follows that:

1. Either $\exists s^{l+1}$ such that $s^l \xrightarrow{a} s^{l+1}$ and $(t^{l+1}, s^{l+1}) \in R$. This implies that $\delta_\Omega(s^l, a) \neq \emptyset$.

2. Or $\exists \alpha$. $s^l \xrightarrow{\alpha|a} s^{l+1}$ and $(t^{l+1}, s^{l+1}) \in R'$ for some $R' \in Z$. Then we choose $\Omega(\sigma(s^l, \tau b), \alpha) = 1$, hence $\delta_\Omega(s^l, a) \supseteq \delta_u(s^l, \alpha \mid a) \odot \Omega(\sigma(s^l, \tau b), \alpha) \neq \emptyset$.

Now we check the reverse direction. Let $\delta_\Omega(s^l, a) \neq \emptyset$. Then there are two cases:

1. Either $\exists s^{l+1}.s^l \xrightarrow{a} s^{l+1}$ then by the property of Z we have $\exists t^{l+1}.t^l \xrightarrow{a} t^{l+1}$, thus $\delta_t(t^l, a) \neq \emptyset$, and $(t^{l+1}, s^{l+1}) \in R$.

2. Or $\exists \alpha, s^{l+1}.s^l \xrightarrow{\alpha|a} s^{l+1}$ and $\Omega(\sigma(s^l, \tau), \alpha) = 1$. The orchestrator $\Omega(\sigma(s^l), \alpha)$ is set to 1 only if $\exists b \in \Sigma$ such that $t^l \xrightarrow{b} t'$ and $s^l \xrightarrow{\alpha|b} s'$ with $(t', s') \in R'$ for some $R' \in Z$. But from the property of R we know that this is done in such a way that for all $c \in \Sigma$ with $s^l \xrightarrow{\alpha|c} v$, $\exists u.t^l \xrightarrow{c} u \wedge (u, v) \in R$. In particular $s^l \xrightarrow{\alpha|a} s^{l+1} \Rightarrow \exists t^{l+1}.t^l \xrightarrow{a} t^{l+1} \wedge (t^{l+1}, s^{l+1}) \in R'$, thus $\delta_t(t^l, a) \neq \emptyset$.

Combining both results we obtain that for all traces $\tau \in \Sigma^*$ $\delta_t(t, a) \neq \emptyset \Leftrightarrow \delta_\Omega(s, a) \neq \emptyset$ for all $s \in \Delta_\Omega(s^0, \tau)$ and $t^l \in \Delta_t(t^0, \tau)$.

($\Leftarrow$) Let Z be the set of observable sets of the community. Recall that $Z = \{R_\theta \mid \theta \in \Theta\}$ and

$$R_\theta = \{(t, s) \mid \forall x \in L(\theta) \exists \tau \in \Sigma^*.s \in \Delta_\Omega(s^0, \tau) \wedge t \in \Delta_t(t^0, \tau) \\ \wedge \sigma(s, \tau) = x\}$$

Next we show that if for all traces $\tau \in \Sigma^*$, $s \in \Delta_\Omega(s^0, \tau)$, $t \in \Delta_t(t^0, \tau)$, we have $\delta_t(t, a) \neq \emptyset \Leftrightarrow \delta_\Omega(s, a) \neq \emptyset$ then it implies that $\mathcal{S}$ is controllable with respect to $\mathcal{S}_t$. This is done by showing that the set of relations Z has the properties of observation relations as given in definition 2.4.5.

Let $(t, s) \in R_\theta$ for some R_θ. Suppose that $t \xrightarrow{a} t'$ then $\delta_t(t, a) \neq \emptyset$ and it follows by assumption that $\delta_\Omega(s, a) \neq \emptyset$. Therefore $\exists s'$ with :

1. Either $s \xrightarrow{a} s'$. Using the fact that $(t, s) \in R_\theta$ we get that for all $x \in L(\theta)$ $\exists \tau a$ such that

$$s' \in \Delta_\Omega(s^0, \tau a) \text{ because } s \in \Delta_\Omega(s^0, \tau) \wedge s \xrightarrow{a} s'$$
$$t' \in \Delta_t(t^0, \tau a) \text{ because } t \in \Delta_t(t^0, \tau) \wedge t \xrightarrow{a} t'$$
$$\sigma(s', \tau a) = x \text{ because } \sigma(s, \tau) = x \wedge s \xrightarrow{a} s'$$

   therefore $(t', s') \in R_\theta$

2. Or $\exists \alpha \in Com$ such that $\delta_u(s, \alpha \,|\, a) \neq \emptyset$ and $\Omega(\sigma(s, \tau), \alpha) = 1$. Then for all $x\alpha \in L(\theta\alpha) \exists \tau a$:

$$s' \in \Delta_\Omega(s^0, \tau a) \text{ because } s \in \Delta_\Omega(s^0, \tau) \wedge s \xrightarrow{\alpha|a} s'$$
$$t' \in \Delta_t(t^0, \tau a) \text{ because } t \in \Delta_t(t^0, \tau) \wedge t \xrightarrow{a} t'$$
$$\sigma(s', \tau a) = x\alpha \text{ because } \sigma(s, \tau) = x \wedge s \xrightarrow{\alpha|a} s'$$

Therefore $(t', s') \in R_{\theta\alpha}$.
Furthermore, let $(u, v) \in R_\theta$ with $\delta_u(u, \alpha \mid b) \neq \emptyset$ for some $b \in \Sigma$. By the definition of R_θ we know that for all $x \in L(\theta)$ we have $\exists b^1 \dots b^k.u \in \Delta_\Omega(s^0, b^1 \dots b^k) \wedge v \in \Delta_t(t^0, b^1 \dots b^k)$ and $\sigma(s, b^1 \dots b^k) = x$.

Now the fact that $\Omega(x, \alpha) = 1$ coupled with $\delta_u(u, \alpha \mid b) \neq \emptyset$ implies that $\delta_\Omega(u, b) \neq \emptyset$. By the assumption of orchestrator existence it follows that $\delta_t(v, b) \neq \emptyset$. Let $u' \in \delta_\Omega(u, b)$ and $t' \in \delta_t(t, b)$. From the definition of Δ it follows that $u' \in \Delta_\Omega(s^0, b^1 \dots b^k b)$ and $v' \in \Delta_t(t^0, b^1 \dots b^k b)$. In addition $\sigma(b^1 \dots b^k) = x$ and $u \xrightarrow{\alpha|b} u'$ implies that $\sigma(u', b^1 \dots b^k b) = x\alpha$. Collecting all these results we get that for all $x\alpha \in L(\theta\alpha)$ $\exists b^1 \dots b^k b.u' \in \Delta(s^0, b^1 \dots b^k b) \wedge$ $v' \in \Delta(t^0, b^1 \dots b^k b) \wedge \sigma(u', b^1 \dots b^k b) = x\alpha$ therefore $(u', v') \in R_{\theta\alpha}$

From the above we deduce that Z is a set of observation relations and therefore $\mathcal{S}$ is controllable with respect to $\mathcal{S}_t$. □

2.5 Communicating services

Thus far all the services we have considered do not communicate with each other. Most interesting applications of services, however, require some form of communication or at least some mechanism whereby the result of one action by some services can be used as an input by some other service. We extend our model with an additional LTS called an *environment*. The environment interacts with all the services and thus allows them to communicate. An action by a service can change the state of the environment. Conversely some actions of the services can depend on the state of the environment, being able to be fired only in some specific environment state. The environment not only represent an abstraction of the how the data is transferred but also it could represent the physical world when one considers physical agents as service providers.

Before proceeding with the required formal definitions it would be helpful to illustrate the aforementioned ideas using a simple example.

Example 2.5.1. *The community involves two services where each can perform a* ***search****,* ***buy*** *and* ***pay*** *actions. The* ***search*** *action is used to located books using appropriate keywords. The service that performs the* ***search*** *action acts as an agent that searches through different online bookstores and returns the result of the search. The* ***buy*** *action is a commitment by the user to buy a book, say putting it in a chopping cart. For simplicity there is no option to remove a book from the cart. Finally the* ***pay*** *action is to pay for the book and in case the cart is empty it just terminates the transaction. The result of* ***search*** *action includes enough information so that a given book could be located by one service and bought by some other service. Similarly we assume that one can use the payment service of one service to pay for the chopping cart from another. This is done by assuming that there is some hidden "transfer cart content" action which allows the content of the cart to be send from one service to another.*

To be able to accomplish these tasks it is clear that the services should be able to communicate or at least the results of one actions to be taken as input for other actions. Towards that end we

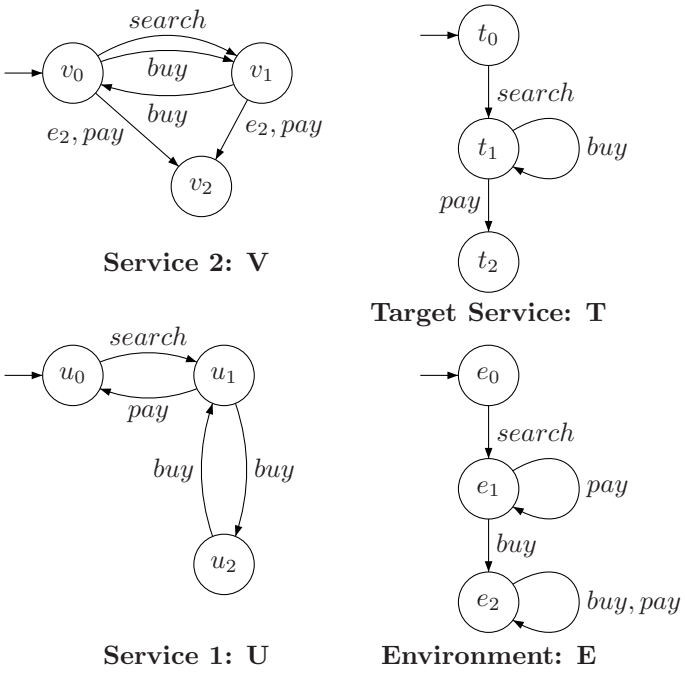


Figure 2.14: Example setup: U and V are available services, E the environment and T the target service.

introduce an environment where actions modify the state of the environment and results are also stored in the environment.

The LTS of services U, V, the environment E and the target T shown in Figure 2.14. In this model the evolution of any services is dictated or constrained by the environment. This is done by forcing services and the environment to evolve in a ***synchronous*** *manner. In the example in Figure 2.14, the environment enforces the requirement that a* ***search*** *has to be done first. Also it is used as a data box where results of* ***search*** *and* ***buy*** *are saved. The first service U can do all three actions:* ***search***, ***buy*** *and* ***pay***. *Service U offers a kind of buy one and get one for half the price service, therefore a client has to buy items in pairs before paying. The second service V can* ***search***, ***buy***, *and* ***pay*** *for items. The service V forces at least one* ***buy*** *after a* ***search*** *because the pay transition can be executed only if the environment is in state e_2. To reach that state at least one item needs to be bought. This is indicated by the label e_2, pay in the* ***pay*** *transition, meaning that this transition can be done only when the environment is in state e_2, i.e. at least one item was purchased. It should be mentioned that transitions not labelled explicitly with an environment state are actually implicitly labelled by* ***all*** *states.*

Having the above example in mind we define formally the environment as an LTS.

Definition 2.5.1. *En environment $\mathcal{E}$ is a tuple $\mathcal{E} = \langle E, \Sigma, e^0, \delta_E \rangle$ where*

- E *is a finite set of states.*
- e^0 *is the initial state of the environment.*
- Σ *is the set of actions.*
- $\delta_E \subset E \times \Sigma \times E$ *is the transition relation.*

As before we write $(e_1, a, e_2) \in \delta_E$ *as* $e_1 \xrightarrow{a} e_2$ *and we define the set* $\delta_E(e_1, a) = \{e_2 \mid e_1 \xrightarrow{a} e_2\}$.

The environment imposes constraints on the behavior of services. For example some actions by a given service can be performed only after it received a certain message or some other event occured. This conditional action is modeled by the environment state. Next we give the formal definition of services in the presence of the environment.

Definition 2.5.2. *An available service* $\mathcal{S}_i$ *over an environment* $\mathcal{E}$ *is a tuple* $\mathcal{S}_i = \langle S_i, \Sigma_i, Com_i, s_i^0, G_i, \delta_i \rangle$ *where*

- S_i *is a finite set of states.*
- Σ_i *is the action alphabet and* Com_i *is the set of communication messages as before.*
- s_i^0 *is the initial state.*
- G_i *is a set of constraint functions of the form* $g : E \to \{\mathbf{true}, \mathbf{false}\}$.
- $\delta_i \subseteq S_i \times G \times (\Sigma_i \cup Com_i \times \Sigma_i) \times S_i$.

The above definition makes the allowed transitions of a service dependent on the environment state. We write $(s_1, g, a, s_2) \in \delta_i$ as $s_1 \xrightarrow{g,a} s_2$. As before it is convenient to use a functional notation. The functional form of the transition is related to the transition relation as follows: $s' \in \delta_i(s, e, a)$ iff $\exists g \in G$ such that $(s, g, s', a) \in \delta_i$ and $g(e) = \mathbf{true}$. Because of its convenience, it is useful to give the formal definition of the functional notation.

$$\delta_i(s, e, a) = \{s' \in S_i \mid (s, g, a, s') \in \delta_i \wedge g(e) = \mathbf{true} \text{ for some g } \in G_i\} \tag{2.2}$$

As an example we define the service V introduced in Figure 2.14 using the general notation.

$$V = \langle \{v_0, v_1, v_2\}, (\Sigma_i \cup Com_i \times \Sigma_i), v_0, G, \delta_v \rangle$$

Where

- $\Sigma_i = \{search, buy, pay\}$, $Com_i = \{1\}$.
- $G = \{True, isBought\}$ where $True(e) = \mathbf{true}$ for all $e \in \mathcal{E}$ and $isBought(e) = \mathbf{true}$ iff $e = e_2$.

- And the transition relation δ_v is given by

$$\delta_v = \{(v_0, True, search, v_1), (v_0, True, buy, v_1), \\ (v_1, True, buy, v_0), (v_0, isBought, pay, v_2), (v_1, isBought, pay, v_2)\}$$

Because both services have access to the environment where they can store and retrieve data, it is possible for one service to search for books, store the results in the environment, and the other service will read the result of the search from the environment to buy items. Similarly, it is possible for one service to buy items and the other to pay for them. The following definition are very similar to the case without an environment.

Definition 2.5.3 (Community of Services With Environment)**.** *A community of n available services* $\mathcal{S}_i = \langle S_i, \Sigma_i, Com_i, s_i^0, G_i, \delta_i \rangle$*,* $i = 1 \dots n$*, is the tuple* $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, G, \delta_u \rangle$ *where*

- $S = S_1 \times \ldots \times S_n$.
- $s^0 = (s_0^1, \ldots s_0^n)$.
- $\Sigma_u = \cup_i \Sigma_i$
- $Com = \cup_i Com_i$
- $G = \cup_i G_i$
- $\delta_u \subseteq S \times (\Sigma \cup Com \times \Sigma) \times S$

The transition relation δ_u is the asynchronous product of all relations δ_i defined as

$$(\langle s_1, \ldots, s_n \rangle, \alpha, g, \langle s_1', \ldots, s_n' \rangle) \in \delta_u \text{ iff } (s_k, \alpha, g, s_k') \in \delta_k \text{ for some } 1 \leq k \leq n \\ \text{and some } g \in G_k \\ \text{and for all } i \neq k \text{ we have } s_i = s_i'$$

The functional notation δ_u remains as before, but it has an extra parameter, the environment state:

$$\delta_u(s_1, \ldots, s_n, e, \alpha) = \bigcup_{i=1}^{n} \bigcup_{s_i' \in \delta_k(s_i, e, \alpha)} (s_1, \ldots, s_i', \ldots, s_n)$$

Similarly to the case without an environment the *orchestrated* transition function is defined as

$$\delta_\Omega(s, e, a) = \delta_u(s, e, a) \bigcup_{m \in Com} \delta_u(s, e, m \mid a) \odot \Omega(m)$$

Where $s = \langle s^1, \ldots, s^n \rangle$ is a community state and the orchestrator Ω at this point is under specified. But also , the environment restricts the transition of a service in two ways. The first is by making the transition dependent on the environment state. This is taken care off in the definition of the transition function, e.g. $\delta_i(s, e, a)$. The second is the condition that for action, say a, to be performed by the service, the environment should also be able to perform an a transition. We illustrate by using the example shown in Figure 2.14. Consider, as an example, the case when service V is in state v_0 and the environment is in state e_0. Even though in principle V can make transition $v_0 \xrightarrow{buy} v_1$, it cannot actually do it because the environment has no transition $e_0 \xrightarrow{buy}$. Therefore the transition of the services in the presence of the environment is basically the *synchronous* product of both transitions. We formalize the above by introducing a new transition functions $\hat{\delta}_\Omega : S \times E \times \Sigma \longrightarrow S \times E$ and $\delta_t : S_t \times E \times \Sigma \longrightarrow S_t \times E$:

$$\hat{\delta}_\Omega(s, e, a) = \delta_\Omega(s, e, a) \times \delta_E(e, a)$$

Similarly the transition function of the target is is given by

$$\hat{\delta}_t(t, e, a) = \delta_t(t, e, a) \times \delta_E(e, a)$$

Finally, the multi-step transition function is defined, in a recursive manner, as in the case without environment. Let $\tau \in \Sigma^*$ then:

$$\Delta_\Omega(s^0, e^0, \tau a) = \bigcup_{(s,e) \in \Delta(s^0, e^0, \tau)} \hat{\delta}_\Omega(s, e, a)$$
$$\Delta_t(t^0, e^0, \tau a) = \bigcup_{(t,e) \in \Delta(t^0, e^0, \tau)} \hat{\delta}_t(t, e, a)$$

Before giving the formal composition characterization we mention a property that will be useful later. Namely, that if the target starts initially from some environment state e_0, and reaches, after a sequence of transitions $\tau \in \Sigma^*$, a set of environment states, then the community will reach the same set of environment states if it starts from the same state e_0 and performs the same sequence of transitions τ. It is useful to formalize the previous statement with some notations. Let $A \subseteq S_t \times E$ be a set of pairs (t, e) denoting a target state in the presence of an environment. Similarly, $B \subseteq S \times E$ is a set of pairs (s, e) denoting a community state in the presence of an environment. Define the function $\lambda : Q \times E \to E$ by the operation $\lambda(t, e) = e$ to extract the environment state, where Q could refer to S or S_t. Below we show that if $\lambda(A) = \lambda(B)$ then for all $a \in \Sigma$ we have $\lambda(\hat{\delta}_t(A, a)) = \lambda(\hat{\delta}_\Omega(B, b))$.

$$
\begin{aligned}
\lambda(\hat{\delta}_t(A,a)) &= \lambda\left(\bigcup_{(t,e)\in A}\hat{\delta}_t(t,e,a)\right)\\
&= \lambda\left(\bigcup_{(t,e)\in A}\delta_t(t,e,a)\times\delta_E(e,a)\right)\\
&= \bigcup_{(t,e)\in A}\lambda\left(\delta_t(t,e,a)\times\delta_E(e,a)\right)\\
&= \bigcup_{(t,e)\in A}\delta_E(e,a) = \bigcup_{e\in\lambda(A)}\delta_E(e,a)\\
&= \bigcup_{e\in\lambda(B)}\delta_E(e,a) \text{ because by assumption } \lambda(A)=\lambda(B)\\
&= \lambda(\hat{\delta}_\Omega(B,a)) \qquad (2.3)
\end{aligned}
$$

In the above we have assume that $\delta_\Omega(s,e,a)\neq\emptyset$ and $\delta_t(t,e,a)\neq\emptyset$. By using equation (2.3) inductively we can easily show that for any arbitrary trace we have:

$$\lambda(\Delta_\Omega((s^0,e^0),\tau)) = \lambda(\Delta_t((t^0,e^0),\tau))$$

2.5.1 Orchestrator with perfect information

The role of the orchestrator in the presence of the environment is the same as in its absence. For any arbitrary history the community should be able to perform the same actions as the target and *only* those actions. In this section we present the orchestrator in the case of perfect information. As before we define the orchestrator formally.

Definition 2.5.4 (Orchestrator with perfect information in the presence of environment)**.** *Given a community of n services, $\mathcal{S}=\langle S,\Sigma_u,Com,s^0,G,\delta_u\rangle$, an orchestrator with perfect information is a function $\Omega: Com\times S\times\Sigma^*\longrightarrow\{0,1\}$.*

Note that from the above definition, the orchestrator does not depend explicitly on the environment.

Definition 2.5.5 (Behavior Composition With Perfect Information With Environment)**.** *Let $\mathcal{S}_t$ be a deterministic target service and $\mathcal{S}$ be community of n available services in the presence of environment. Let Ω be an orchestrator with perfect information and denote by $\mathcal{S}_\Omega$ the orchestrated community. We say that $\mathcal{S}_\Omega$ is a behavior composition of $\mathcal{S}_t$ iff for all traces $\tau\in\Sigma^*$ and all target states $(t,e)\in\Delta_t(t^0,\tau)$ and for all $(s,e)\in\Delta_\Omega(s^0,\tau)$ we have*

$$\forall a\in\Sigma,\quad \hat{\delta}_t(t,a)\neq\emptyset\Leftrightarrow\hat{\delta}_\Omega(s,a)\neq\emptyset$$

First note that the condition $\hat{\delta}_t(t,e,a) \neq \emptyset \Leftrightarrow \hat{\delta}_\Omega(s,e,a) \neq \emptyset$ implies that $\delta_t(t,e,a) \neq \emptyset \Leftrightarrow \delta_\Omega(s,e,a) \neq \emptyset$. Second, in the above definition we always compare the target and the system in the *same* environment state. This is possible because we have already shown that starting from the same state e^0 and performing the same sequence of transitions τ both the target and the community will be in the same set of environment states.

The characterization of the problem in the presence of the environment is very similar to the case when the environment was absent.

Definition 2.5.6. *A relation $R \subseteq S_t \times E \times S$ is said to be a controllability relation iff for all $(t,e,s) \in R$ and all actions $a \in \Sigma$*

1. *If $t \xrightarrow{a} t'$ and $\delta_E(e,a) \neq \emptyset$ then*

   (a) *Either $\exists s'.s \xrightarrow{a} s' \wedge (t',e',s') \in R,\ \forall e' \in \delta_E(e,a)$*

   (b) *Or $\exists \alpha, s'$ such that $s \xrightarrow{\alpha|a} s'$ and for all $b \in \Sigma$*

$$(s \xrightarrow{\alpha|b} s'' \wedge \delta_E(e,b) \neq \emptyset) \Rightarrow \left(\exists t''.t \xrightarrow{b} t'' \wedge (t'',e'',s'') \in R \text{ for all } e'' \in \delta_E(e,b)\right)$$

2. *If $s \xrightarrow{a} s'$ and $\delta_E(e,a) \neq \emptyset$ then $\exists t'.t \xrightarrow{a} t' \wedge (t',e',s') \in R$ for all $e' \in \delta_E(e,a)$*

this is a straightforward generalization of the previous definition but including the environment. In the above definition, condition 1b takes care of "side effects", it makes sure that if the orchestrator enables an "a"-action via some message α, all other actions "b" that are also enabled by message α lead to states that are also in R. Before giving the theorem that relates the existence of an orchestrator to that of an controllability we prove a useful lemma, similar to lemma 2.3.3 in the case when the environment was absent.

Lemma 2.5.2. *Let $R \subseteq S_t \times E \times S$ be an controllability between a target $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, \delta_t \rangle$ and a community $\mathcal{S} = \langle S, \Sigma_u, s^0, G, \delta_u \rangle$ in the presence of environment $\mathcal{E} = \langle E, \Sigma, e^0, \delta_E \rangle$. Then there exists an orchestrator with perfect information Ω such that for all $(t,e,s) \in R$, $t \in S_t$,$s \in S$, $e \in E$, and for all $a \in \Sigma = \Sigma_u \cup \Sigma_t$ we have $\hat{\delta}_t(t,e,a) \neq \emptyset \Leftrightarrow \hat{\delta}_\Omega(s,e,a) \neq \emptyset$. Furthermore for all $t' \in \delta_t(t,e,a)$, $s' \in \delta_\Omega(s,e,a)$ and $e' \in \delta_E(e,a)$ we have $(t',e',s') \in R$.*

Note that in the above lemma the possible states of the environment produces by $\hat{\delta}_t(t,e,a)$ and $\hat{\delta}_\Omega(s,e,a)$, is the same as we have shown previously.

Proof. The proof is similar to the proof of lemma 2.3.3. The detailed proof is given in the appendix.

□

The next theorem links the existence of an orchestrator to the existence of an controllability as in the case when the environment was absent.

Theorem 2.5.3. *Given a target service $\mathcal{S}_t = \langle S_t, \Sigma_t, s_t^0, G, \delta_t \rangle$ and a community of n available services $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, G, \delta_u \rangle$, then an orchestrator with perfect information Ω exists such that $\mathcal{S}_\Omega$ is a composition iff $\mathcal{S}$ is controllable with respect to $\mathcal{S}_t$.*

Proof. Similar to the proof of theorem 2.3.4. The detailed proof is given in the appendix. □

2.5.2 Orchestrator with partial information

In this section we develop the theory for the case of an orchestrator with partial information in the presence of the environment (we use the shorthand *i.t.p.* for "in the presence"). The concepts and terminology are very similar to the case when the environment is absent as presented in section 2.4. Also it differs from the previous section, the case with perfect information by the information available to the orchestrator. In this case the orchestrator is a function of the "message trace" of the state. This message trace is computed via the σ function introduced in section 2.4. Then given a state of the community s reached from the initial state s^0 after a sequence τ of actions then the orchestrator is $\Omega(\sigma(s,\tau),\alpha)$ where $\alpha \in Com$ is a communication message and the message trace function σ was defined in section 2.4. As in the case of perfect information, the *orchestrated* transition function is given as:

$$\delta_\Omega(s,e,a) = \delta_u(s,e,a) \bigcup_{\alpha \in Com} \delta_u(s,e,\alpha \,|\, a) \odot \Omega\left(\sigma(s,\tau),\alpha\right)$$

Note that the value all the definitions above are exactly the same as in the case of perfect information presented in 2.5.1, except the dependence of the orchestrator Ω on the message trace part of the trace.

The behavior composition problem is exactly as the perfect information case and given in definition 2.5.5.

Also in this case the existence of an orchestrator is characterized by the existence of a set of observation relations with the addition of the environment. Formally,

Definition 2.5.7 (Observation relations *i.t.p.* of environment)**.** *A set of relations $Z \subseteq 2^{S_t \times E \times S}$ is called a set of observation relationsi.t.p. of environment $\mathcal{E}$ iff : for every $R \in Z$ and for every $(t,e,s) \in R$ we have the following*

1. *If $s \xrightarrow{g(e),a} s' \wedge \delta_E(e,a) \neq \emptyset$ then $\exists t'.t \xrightarrow{g(e),a} t' \wedge (t',e',s') \in R$*
2. *If $t \xrightarrow{g(e),a} t' \wedge \delta_E(e,a) \neq \emptyset$ then*
   (a) *Either $\exists s'.s \xrightarrow{g(e),a} s' \wedge (t',e',s') \in R$*

(b) Or $\exists s', \alpha \in Com, R' \in Z$ *with*

$$s \xrightarrow{g(e),\alpha|a} s' \wedge (t', e', s') \in R'$$
$$\wedge$$
$$\forall (u, v, w) \in R(w \xrightarrow{g(v),\alpha|b} w' \Rightarrow \exists u'.u \xrightarrow{g(v)|b} u' \wedge (u', v', w') \in R')$$

Definition 2.5.8 (Controllability under partial information *i.t.p.* of environment)**.** *A community of services* $\mathcal{S}$ *is said to be controllable under partial information with respect to a target service* $\mathcal{S}_t$ *i.t.p of environment* $\mathcal{E}$ *iff there exists a set of observation relations* $Z \subseteq 2^{S_t \times E \times S}$ *such that* $\exists R_0 \in Z$ *with* $(t^0, e^0, s^0) \in R_0$*.*

The above definition allows us to characterize the existence of an orchestrator in a manner similar to the case of perfect information.

Theorem 2.5.4. *Let* $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ *be a community of services,* $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, \delta_t \rangle$ *be a target service, and* $\mathcal{E} = \langle E, \Sigma, e^0, \delta_E \rangle$ *be an environment. An orchestrator with partial information,* Ω*, exists such that* $\mathcal{S}_\Omega$ *is a composition of* $\mathcal{S}_t$ *iff* $\mathcal{S}$ *is controllable under partial information with respect to* $\mathcal{S}_t$ *i.t.p of environment* $\mathcal{E}$*.*

Proof. Similar to the proof of theorem 2.4.1. The detailed proof is given in the appendix. □

2.6 Conclusion

In this chapter the basic framework for the behavior composition was presented. We have also proved some general results that are part of this thesis contribution to the behavior composition problem. The first result is the use of the concept of controllability relation to prove that for a given community $\mathcal{S}$ and target $\mathcal{S}_t$ an orchestrator exists such that the orchestrated community $\mathcal{S}_\Omega$ is a behavior composition if and only if $\mathcal{S}$ is controllable with respect to $\mathcal{S}_t$. We also proved this property for the case when the specification is expressed using the richer and less restrictive modal specifications. We also introduced the concept of the set of observation relations and we proved that an orchestrator with partial observation exists if and only if a set of observation relations exists between the community and the target. It was also shown that behavior composition problem is related to the classical control problem. Finally, a similar study was performed for the case when an environment is present. This chapter related the existence of, and the construction of, an orchestrator to the existence and construction of the controllability relation (set of observation relations in the partial information case). The next chapters will use these results and present methods to efficiently find the controllabilty relation, or the observation relations in the case of partial information.

Chapter 3

Orchestration under perfect information

Contents

3.1 Introduction

In this chapter we study the composition problem when the orchestrator has perfect information. This means that the orchestrator knows exactly in which state the community is in. This is done by using the fact, proved in chapter 2 that an orchestrator exists if and only if a controllability relation exists between the community and the target. We develop an efficient on-the-fly algorithm to find a controllability relation when on exists.

Before proceeding any further, it is useful to recall some needed definitions from chapter 2. Let $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ and $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, \delta_t \rangle$ be the community and target service respectively. Define the common action alphabet $\Sigma = \Sigma_u \cup \Sigma_t$. Then the *orchestrator under perfect information* is a function

$$\Omega : S \times \Sigma^* \times Com \longrightarrow \{0, 1\}$$

Recall that the *orchestrated community*, or the community controlled by Ω, is the tuple $\mathcal{S}_\Omega = \langle S, \Sigma, s^0, \delta_\Omega \rangle$ where

$$\delta_\Omega(s, a) = \delta_u(s, a) \bigcup_{\alpha \in Com} \delta_u(s, \alpha) \odot \Omega(s, \tau, \alpha) \tag{3.1}$$

where $\tau \in \Sigma^*$ is the sequence of actions performed to reach s from the initial state s^0. Clearly τ is not unique but the function δ_Ω is not used by itself. Rather is used as a short hand to compute the states the community could be in, after a sequence of actions τ. This is done via the extended function of the *orchestrated community* which is defined recursively:

$$\begin{aligned} \Delta_\Omega(s^0, \epsilon) &= \{s\} \\ \Delta_\Omega(s^0, \tau a) &= \bigcup_{s' \in \Delta(s^0, \tau)} \delta_\Omega(s', a) \end{aligned}$$

Replacing the definition of δ_Ω in equation (3.1) in the above we get

$$\begin{aligned} \Delta_\Omega(s^0, \epsilon) &= \{s\} \\ \Delta_\Omega(s^0, \tau a) &= \bigcup_{s' \in \Delta(s^0, \tau)} \left[\delta_u(s', a) \bigcup_{\alpha \in Com} \delta_u(s', \alpha \,|\, a) \odot \Omega(s, \tau, \alpha) \right] \end{aligned}$$

The above definition shows the context in which the sequence of actions τ is used in. In this context there is no ambiguity in its usage. Intuitively $\mathcal{S}_\Omega$ is a composition of $\mathcal{S}_t$, if after an arbitrary sequence of actions τ, and regardless in which state the community is in, the community should be able to perform any action that can be performed by the target and *only* those actions. Now since the target is deterministic, and the composition can perform only the actions that the

target can perform then there is a one-to-one correspondence between the sequences of actions and the states of the target thus we can write:

$$\Omega : S \times S_t \times Com \longrightarrow \{0, 1\}$$

According to the above we can write $\Omega = \Omega(s, t, \alpha)$ where t is the target state reached when the target executes the sequence τ. Finally, in theorem 2.3.4 in chapter 2 we proved that the existence of a composition is tied to the existence of a controllablility relation. We also showed how to compute the values of the orchestrator from the controllability graph. In this chapter we present a novel method, for the case of perfect information, to find the controllability relation, if one exists, and therefore the orchestrator.

A special case of his problem has been solved by different methods including the use of Propositional Dynamic Logic [DGS07], the concept of simulation in the case of deterministic services, and using the concept of ND-simulation in the case of non-deterministic services [SPD08]. The main hurdle to overcome is that the worst-case complexity of the problem is known to be EXPTIME *in the number of services* [MW08]. Therefore it is important to find a heuristic or a method that minimize this complexity in the average case. While the lower bound tell us this is not possible in general it its possible in some cases using some heuristics.

The main aim of this chapter is to develop an efficients algorithm to compute the controllability relation. This is accomplished by developing a new on-the-fly algorithm that has two advantages over existing methods: it searches part of the state space until it finds a solution, if one exists. This is in contrast to fixpoint methods which consider the whole state space and remove non-conforming states until a solution is found. Second the way the algorithm is constructed it is possible to use a heuristic as input to speed up the search. We actually develop such a heuristic by using an abstraction method and we use it as input to the algorithm. While the worst-case complexity of the proposed algorithm is also exponential *in the number of services* (this is a lower bound, see [MW08]) we argue that in the average case it is much better.

This algorithm improves on previous approaches to the problem and advances the state of the art in service composition by proposing an algorithm that: 1) visits states as needed, which allows it to deal efficiently with systems containing a large number of complex services; 2) is self-contained and can be easily incorporated in any other model; 3) is robust with respect to service failure.

The service composition problem we consider here shares similarities with some occurrences of the control problem and can be tackled by similar approaches (see [BCF08a] for example). The service composition problem could also be transformed to be solved by powerful tools used for software verification. However, we believe that service composition, although similar to other problems, has some important specificities. In addition to the possibility of using a heuristic to speed up the search one can add to the algorithm of this chapter considerations about quality of service or quality of experience [EHMR10]: these information about services can be taken into consideration when choosing from a candidate to perform a specific action to match a requirement of the goal. This advocates for developing a specific algorithm that can be enhanced further to account for the particularities of the service composition problem.

The remaining sections of this chapter can be divided into two parts. The first part includes section 3.2 only, discusses existing approaches to the composition problem. The second part includes all remaining sections and detail our contribution. More specifically in section 3.2 we use the special case of the Roman Model to present a common approach in the literature for solution the composition problem which we call the fixpoint approach. In section 3.3 we present our on-the-fly algorithm and discuss its advantages over the fixpoint approach. In section 3.4 we show that our algorithm is correct and we compute its worst case complexity. In section 3.5 we show that our proposed algorithm is robust with respect to service failure. In section 3.6 we present an abstraction method that reduces drastically the state space of the community and the target. We prove that if no simulation relation exists between the abstracted community and the abstracted target then no controllability relation exists between the original system. When a simulation relation exists it is used as a heuristic to speed up the algorithm. In section 3.7 we present the algorithm for our general model presented in chapter 2. The target for general model is specified using modal specifications. We conclude with section 3.8.

3.2 The Roman Model and fixpoint methods

In order to make the case for the proposed on-the-fly algorithm it is important to contrast it with current methods used to solve the composition problem. All current approaches need to do *full state space* computation to reach a solution. In fact, most of them use a fixpoint approach: *start with the full state space* and remove states using some operator that is method dependent, until no more states can be removed, then the remaining states are the solution. In this section we discuss one such approach in order to contrast it with our approach presented in the next section.

The on-the-fly algorithm we are proposing can become unwieldy and a bit complicated. Because of that and as a first step we introduce the algorithm to handle the special case of the Roman Model. As we have seen in Chapter 2 the Roman Model is a special case of the general model we are proposing. At a later stage the algorithm that handles our general model is presented.

3.2.1 Roman Model

It is important to fix the terminology for the problem, which remains largely as presented in the previous chapter, in the special case of the Roman Model. There are n available services $\mathcal{S}_i$ in the presence of an environment $\mathcal{E}$. The target is modelled by target service $\mathcal{S}_t$. It is convenient to handle states of the community as a whole thus a state of the community s, where $s = \langle s_1, \ldots s_n \rangle$ then service $\mathcal{S}_i$ is in state s_i. Recall that the environment is a non deterministic LTS like the services and imposes constraints on the transition of the services. For example, given a service $\mathcal{S}_i$ a transition $s_i \xrightarrow{g(e),a} s_i'$ means that the service can move from s_i to s_i' only if $g(e) = true$. This has the effect that some transitions are allowable only in certain environment states. We

also showed in section 2.5 that for a given history, the composition and the target will be in the *same* environment state.

In the Roman Model all transitions are controllable therefore the transition function of the community is written as

$$\delta_\Omega(s,e,a) = \bigcup_{m \in Com} \delta_u(s,e,m\,|\,a) \odot \Omega(s,x,m)$$

Again, $x \in \Sigma^*$ is the sequence of actions executed to reach state s and as explained in the previous section it should cause no ambiguity when used in the context of the extended transition function. Also, instead of having a set of communicating messages Com, each service can be communicated with using its index. For example the message ka means "instruct service k to execute transition a". Define $\{n\} = \{1,\ldots,n\}$ then $Com = \{n\} \times \Sigma$. Furthermore, for a given message ka one cannot have $s \xrightarrow{ka|b} s'$ with $a \neq b$. Therefore in the Roman Model the community of n services is a tuple $\mathcal{S} = \langle S, \Sigma_u, \{n\}, s^0, \delta_u \rangle$ where it is understood that the set of messages $Com = \{n\} \times \Sigma$ with the above restriction. The transition function of the community becomes:

$$\delta_\Omega(s,e,a) = \bigcup_{k=1}^{n} \delta_u(s,e,ka\,|\,a) \odot \Omega(s,x,ka) \tag{3.2}$$

But since the transition function of the community is the asynchronous product of the transition function of the available services then we can write:

$$\delta_u(s,e,ka\,|\,a) = \bigcup_{s'_k \in \delta_k(s,e,ka|a)} \langle s_1,\ldots,s'_k,\ldots,s_n \rangle$$

Where δ_k is the transition function for service k. Recall also that for a given transition $s \xrightarrow{g(e),a} s'$ even if $g(e) = true$ the community does not make the transition unless $\exists e'$ such that $e \xrightarrow{a} e'$. Basically the transition function of the community in the presence of an environment is the synchronous product of the community transition in the absence of an environment and the transition of the environment. Therefore the transition function of the orchestrated community, in the presence of the environment is given as a modified transition function $\hat{\delta}$ defined by:

$$\hat{\delta}_\Omega(s,e,a) = \delta_\Omega(s,e,a) \times \delta_E(e,a)$$

Similarly the transition function of the target in the presence of the environment is given by

$$\hat{\delta}_t(t,e,a) = \delta_t(t,e,a) \times \delta_E(e,a)$$

Where $\times$ is the Cartesian product of two sets. The definition of the multistep transition for both the community and target is unchanged and still given by

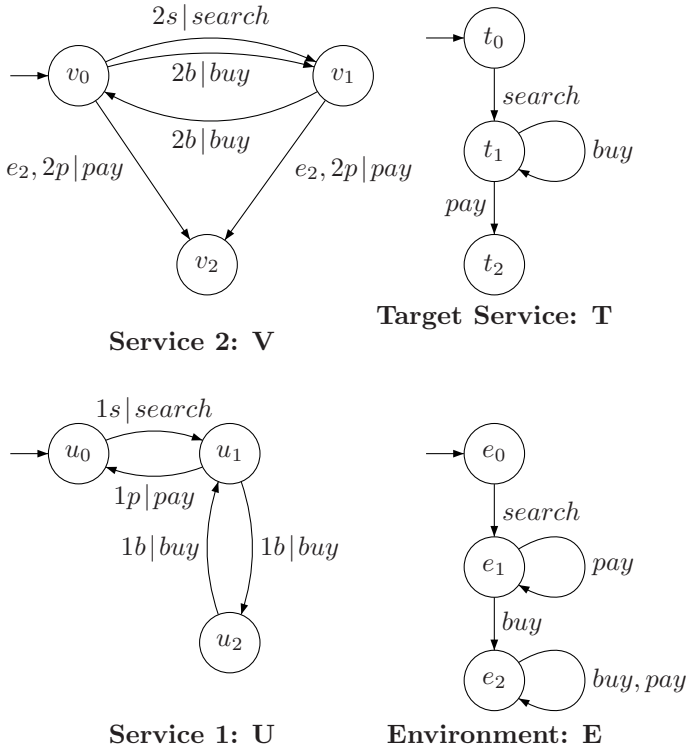


Figure 3.1: Example setup: U and V are available services, E the environment and T the target service. The letters s, b and p are shorthand for search, buy and pay respectively.

$$\Delta_\Omega(s^0, e^0, \tau a) = \bigcup_{(s',e') \in \Delta_S(s^0,e^0,\tau)} \hat{\delta}_\Omega(s', e', a)$$
$$\Delta_t(t^0, e^0, \tau a) = \bigcup_{(t',e') \in \Delta_t(t^0,e^0,\tau)} \hat{\delta}_t(t', e', a)$$

Recall also that the orchestrator, Ω, we are seeking is the one that makes the *orchestrated* community a composition of the target: for any arbitrary sequence of actions, τ and for all $(t, e) \in \Delta_t(t^0, e^0, \tau)$, $(s, e) \in \Delta_\Omega(s^0, e^0, \tau)$ we have, for all actions $a \in \Sigma$

$$\hat{\delta}_t(t, e, a) \neq \emptyset \Leftrightarrow \hat{\delta}_\Omega(s, e, a) \neq \emptyset$$

3.2.2 Example 1

Before presenting the formal setting it is useful to compare the two different approaches and motivate the idea of an on-the-fly algorithm. This is done using the example taken from section

2.5 and for convenience shown again in Figure 3.1. The actions supported by the two services of the community are *search*, *buy* and *pay* which we can shorten by s, b and p respectively. Both services are fully controllable with the first service, U, receiving messages from the set $\{1s, 1b, 1p\}$ and the second service, V, receiving messages from the set $\{2s, 2b, 2p\}$. Since we are in the context of the Roman Model one can infer from the message the action and the service. For example, by sending message $2p$ it is clear that only the "pay" action of the second service would be enabled.

The full state space and four possible solutions, together with the target, are shown in Figures 3.2 and 3.3. Note that in these figures the actions of the community, which can be inferred from the messages, were omitted for readability. For example $(u_0, v_0, e_0) \xrightarrow{1s} (u_0, v_1, e_1)$ is a shorthand for $(u_0, v_0, e_0) \xrightarrow{1s|s} (u_0, v_1, e_1)$.

Given two controllability relations R_1 and R_2 we say that R_1 is larger than R_2 if $R_2 \subseteq R_1$. As will be shown in section 3.2.3, *a largest* controllability relation exists. In this example, one can see that in such a relation the dashed community states are related to the target state (t_0, e_0), the community states with double ovals are related to the state (t_1, e_1), the community states with ovals are related to (t_1, e_2) and *all* community states are related to (t_2, e_1) and (t_2, e_2) since these two target states don't have any transitions. Therefore the *largest* controllability relation contains all tuples of the form (t_1, e_2, s) and (t_2, e_1, s) for all community states s.

On the other hand we show in red four different solutions to the problem. These solutions are also controllability relations between the community and the target. Clearly there are others and, as will be shown in section 3.2.3 the solution computed using the fixpoint approach (see equation (3.5)), is the union of all those solutions. Since in a typical composition scenario what is needed is just one solution, then computing the largest (i.e. all the solutions) is not necessary and wasteful knowing that the problem is EXPTIME hard. In many, if not most, situations what is needed is a single solution, for example one of the solutions shown in Figures 3.2 and 3.3.

The states labeled red in Figures 3.2 and 3.3 represent the controllability relation and the red transitions are the composition, or the "proof" that the relation is indeed a controllability relation. If one starts with (u_0, v_0, e_0) and at each step, and each target transition, tries to match it with a community transition it is possible to obtain a controllability relation in a much smaller number of computations than required to compute the *largest* controllability relation. One can see that in this case a fraction of the state space is visited whereas any algorithm that computes the *largest* controllability relation needs to visit all the state space, which includes testing all the transitions.

The first solution shown in Figure 3.2, labelled solution1, can be obtained, for example, if the user has a preference for service 1, which means that the community will always try to match the target transition using service 1. One can see that there is no need for backtracking, i.e. explore other branches of the state space, as long as the first choice is transition $1s$ not $2s$. This means that in this case there is one chance out of two that the procedure we are proposing will not need any backtracking at all. In those cases the size of the solution is comparable to the size of the *target* rather that the community which is exponential in the number of services.

Suppose that we have some a priori knowledge that the transition $2s$ does not lead to a solution and therefore choose the transition $1s$. In this case the proposed step-by-step method or on-the-fly algorithm will do no backtracking at all. Such information can be obtained for example from some kind of abstraction as will be shown later on in this chapter. Obviously all the aforementioned techniques are heuristics and it is possible that such on-the-fly procedure makes the "worst" decision on every step. The question is: if the algorithm makes the "worst" case decision does it become less efficient than algorithms that compute the *largest* controllability relation?

In section 3.3 we present an algorithm that implements the aforementioned idea, that builds a controllability relation on the fly. It can take as input a heuristic obtained from, say, abstraction or user preference as mentioned previously. Crucially, we will show that in the worst case, i.e. when the algorithms always makes the "worst" decision, its complexity is the same as algorithms that compute the *largest* relation. Therefore in most situations a lot will be gained from using an on-the-fly method.

Before closing this section it is important to contrast the above procedure with on-the-fly algorithms for computing equivalence between two system, e.g. bisimulation [Par81]. In such situations, the on-the-fly algorithm, even though it is incremental, eventually has to visit all states, and tests all transitions, to determine if two system are equivalent [CS01]. Therefore such methods don't have a real advantage over algorithms that compute the *largest* relations. In our case, one is trying to find some kind of equivalence between a target and a "subgraph" or a small portion of a very large, in fact exponential in the number of services, state space, so in the best case the number of computations is proportional to the size of the target rather than the community. When a solution is found there is no need to continue searching the remaining space.

3.2.3 Fixpoint approach

In this section we give one so called fixpoint approach for computing the *largest* controllability relation. It will allow us to contrast this class of approaches with our proposed algorithm.

As we have shown in theorem 2.5.3 in chapter 2 an orchestrator exists iff there is a controllability relation between the states of the target, the environment and the community. It is the same in the case of the Roman Model except that the definition of an controllability relation is simpler because messages are sent to specific services and they have no "side effect" in the sense that a message ka can only enable a-transitions in service k only, whereas in our general model a message α can enable transitions other than a and services other than k. The formal definition is given next.

Definition 3.2.1 (controllability for the Roman Model)**.** *Let $\mathcal{S}_t = \langle S_t, \Sigma_t, s_t^0, \delta_t \rangle$ be a target service and $\mathcal{S} = \langle S, \Sigma_u, \{n\}, s_0, \delta_u \rangle$ be the community of n services. A relation $R \subseteq S_t \times S$ is said to be a controllability relation if for all $(t, s) \in R$ and for all $a \in \Sigma = \Sigma_t \cup \Sigma_u$:*

- $\exists \{m\}_a \subseteq \{n\}$ *such that for all* $k \in \{m\}_a$ *we have* $s \xrightarrow{g(e),ka|a} s' \Rightarrow \exists t'.t \xrightarrow{g(e),a} t'$ *and*

$(t', e', s') \in R$. *The subscript* a *in the set* $\{m\}_a$ *stresses the fact that it depends on* a.

- $t \xrightarrow{g(e),a} t' \wedge e \xrightarrow{a} e' \Rightarrow \exists k \in \{m\}_a, s'.s \xrightarrow{g(e),ka|a} s' \wedge (t', s') \in R$.

Clearly the above definition is much simpler than the general model. All is needed is that for every target transition to be matched by a service that can make the same transition. The only complication, so to speak, is to take care of the non-determinism of the chosen service. It is convenient to write the above definition in a more compact form as:

$$t \xrightarrow{g(e),a} t' \wedge e \xrightarrow{a} e' \Rightarrow (\exists k, s'.s \xrightarrow{g(e),ka|a} s' \wedge \forall s'' \in \delta_u(s, e, ka\,|\,a) \wedge \forall e'' \in \delta_E(e, a), (t', e'', s'') \in R) \tag{3.3}$$

To proceed with the proposed solution it is important first to introduce what we have referred to so far as the *fixpoint approach*, in order to contrast it to our solution. For that reason it is useful to recast the definition of controllability relation as a fixpoint of a monotonic function. Towards that end let $R \subseteq S_t \times E \times S_1 \times \ldots \times S_n$ be an arbitrary relation between the states of the target, environment and community. Let t, e and s be states of the target, environment and community(i.e $s = \langle s_1, \ldots, s_n \rangle$) respectively. Define the function F over the set of such relations as follows:

$$F(R) = \Big\{ (t, e, s) \in R \mid \forall a \quad t \xrightarrow{g(e),a} t' \wedge e \xrightarrow{a} e' \Rightarrow \Big[\exists k, s'.s \xrightarrow{g(e),ka|a} s' \wedge (t', e', s') \in R \quad \wedge \quad \forall s''(s \xrightarrow{g(e),ka|a} s'' \Rightarrow (t', e', s'') \in R) \Big] \Big\} \tag{3.4}$$

The above definition of F means that every transition of the target (allowed by the environment), can be matched by a community transition (again allowed by the environment). Also, when a controlled transition is enabled through some message ka one needs to make sure to take care of "side effects". Namely, that any other action enabled by ka should be matched by the target also. This is taken care of in the last part of the expression.

Similarly to the standard preorder definitions it is easy to see that a relation R is an controllability relation iff $R \subseteq F(R)$ [CS01]. Since F is also a monotonic function therefore by Tarski's fixpoint theorem a largest fixpoint exists and is given by:

$$\sim = \bigcup \{R \mid R \subseteq F(R)\} \tag{3.5}$$

Note that equality 3.5 implies that there could be many controllability relations and $\sim$ is the union of all of them. In a sense $\sim$ contains all the solution of the problem. A typical procedure,

similar to the one for classical equivalences and preorders [CS01], for computing the largest controllability relation $\sim$, as defined in equation (3.5), would be to define the set of relations:

$$\begin{aligned} \sim_0 &= S_t \times E \times S_1 \times \ldots \times S_n \\ \sim_{i+1} &= F(\sim_i) \end{aligned} \tag{3.6}$$

Since the transitions systems under study are finite then there exists a j such that $\sim_j = F(\sim_j)$. The largest fixpoint, $\sim_j$, is the largest controllability relation one is seeking. Henceforth, this procedure for computing the largest controllability relation is referred to as "fixpoint approach". The important point to note is that one always starts with $\sim_0$, which is, being the product of all the states, exponential in the number of services. This means one always has to visit all the states in $\sim_0$, the full state space, and more importantly, process all transitions. Clearly this is an expensive operation and, as will be shown later, unnecessary. In [GPS13], the authors use a concept, called ND-simulation, similar to controllability for the special case of the Roman Model. They show that after obtaining this non-deterministic simulation, or ND-simulation, one can generate all compositions using what is called orchestrator generator. This not really necessary since what is usually needed is one orchestrator. It is worth mentioning that other methods for solving the composition problem, e.g. [SP09], without the use of the concept of ND-simulation relation also start from the full state space and remove, one by one, non matching states to obtain a solution. In this respect they also are considered "fixpoint approach" algorithms.

3.3 On-the-Fly algorithm for the Roman Model

In the previous section we have presented one "fixpoint" algorithm used to compute the controllability relation. This algorithm is representative of a class of similar approaches, where all start with the full state space and then remove states until a fixpoint is reached.

In this section we present our first contribution in this chapter: an on-the-fly algorithm to find a controllability relation, if one exists. As we have mentioned before, to make the discussion simple we first present the algorithm for Roman Model which is a special case of our model. In section 3.7 we present the algorithm for the general model.

3.3.1 The algorithm

Let S be the set of states of the community of available services , E be the set of environment states and S_t the set of states of the target service. The algorithm maintains two relations $\mathcal{A}$ and $\mathcal{B}$, both initially empty.

The relation $\mathcal{A} \subseteq S_t \times E \times S$, represents the controllability relation that the algorithm is trying to find between the states of the community and the target. Note that there might be more than one controllability relation.

During the execution of the algorithm tuples are added and *removed* from $\mathcal{A}$. The second relation, $\mathcal{B} \subseteq S_t \times E \times S$ represents the set of tuples that were found by the algorithm to be **not**

in the controllability relation. Once a tuple is found to be not in the controllability relation it cannot become in the controllability relation at some later stage *because tuples are added to* $\mathcal{B}$ *but never removed*. The set $\mathcal{B}$ is maintained so that a given tuple is not processed more than once. The algorithm is composed of two mutually recursive functions $CONTROL$ and $MATCH$ that are described next.

Algorithm 1: function CONTROL for the Roman Model case

```
1  CONTROL(t, e, s)
2  if ⟨t, e, s⟩ ∈ B then
3  |  return false
4  if ⟨t, e, s⟩ ∈ A then
5  |  return true
   /* Assume that (t,e,s) are in controllability relation                    */
6  A = A ∪ ⟨t, e, s⟩
7  res = true
   /* Check if controller can send messages to match all the transitions of
      the target                                                             */
8  foreach a ∈ Σ do
9  |  foreach t --g(e),a--> t' ∧ e --a--> e' do
10 |  |  res=MATCH (s, e --a--> e', t --a--> t')
11 |  |  if res = false then
12 |  |  |  Goto Exit
13 Exit:
14 if res=false then
15 |  B = B ∪ ⟨t, e, s⟩
16 |  A = A − ⟨t, e, s⟩
17 |  changed = true
18 return res
```

Function CONTROL. Given a target state (t, e), and a community state (s, e) the function $CONTROL(t, e, s)$ returns true iff the tuple $\langle t, e, s\rangle$ belongs to the controllability relation. Basically, $CONTROL$ performs a depth-first search over the state space. When a tuple is visited for the first time, i.e. not in $\mathcal{A}$ nor in $\mathcal{B}$, it is assumed to be in the controllability relation, and therefore added to $\mathcal{A}$ (lines 2-6). Then the tuple is processed by checking that every transition of the target can be matched by a transition of the community (lines 8-12). It should be noted that since all the community transitions are controllable by the orchestrator checking the reverse is not necessary because any transition that is not enabled explicitly by the orchestrator is implicitly disabled.

Algorithm 2: function MATCH for the Roman Model case

```
1  MATCH(s, e -a-> e', t -a-> t')
   /* s_i is the i^th component of s,i.e.  s = ⟨s_1, ..., s_n⟩ similarly for s' and s''
      */
2  for i = 1 to n do
3  |  foreach s_i -g,a-> s'_i ∧ g_i(e) = true do
4  |  |  ENQUEUE (Q,i,s'_i)
5  res = false
6  while Q ≠ ∅ ∧ res = false do
7  |  s'_k = DEQUEUE(Q)
8  |  res=CONTROL(t', e', s')
9  |  if res=false then
10 |  |  Goto Label
11 |  else
12 |  |  foreach s_k -a-> s''_k do
13 |  |  |  res=CONTROL(t', e', s'')
14 |  |  |  if res = false then
15 |  |  |  |  Goto Label
16 |  Label:
17 return res
```

After a tuple is processed, if it is found to be not in the controllability relation, then it is removed from $\mathcal{A}$ and added to $\mathcal{B}$ (lines 14-17). Note that $CONTROL$ visits (i.e. adds to $\mathcal{A}$) tuples in *preorder* and processes them in *postorder*. We will have more to say about this later.

Given a target state (t, e) and a community state (s, e), the function $CONTROL$ tests whether (t, e, s) is in the controllability relation. This is the case iff for every possible transition of the target state $(t, e) \xrightarrow{a} (t', e')$ the community can match it with an a-transition $(s, e) \xrightarrow{a} (s', e')$ such that (t', e', s') is in the controllability relation (lines 8-12 in $CONTROL$).

Recall that a community state (s, e) is composed of the states of n services, i.e. $s = \langle s_1, \ldots, s_k, \ldots, s_n \rangle$. Then for a given transition $(t, e) \xrightarrow{a} (t', e')$, the algorithm needs to satisfy clause (3.3) by finding a service k such that:

1. (P1) There **exists one** s'_k with $s_k \xrightarrow{g,a} s'_k$ and $g(e) = true$ for some $g \in G_k$ such that (t', e', s') is in the controllability relation, where $s' = (s_1, \ldots, s'_k, \ldots, s_n)$.

2. (P2) For **every** s''_k, such that $s_k \xrightarrow{g,a} s''_k$, it is the case that (t', e', e'') is in the controllability relation, where $s'' = (s_1, \ldots, s''_k, \ldots, s_n)$

The above two conditions are implemented in function MATCH explained below.

Function MATCH. For every target transition, this function tries to find a community transition that matches it. Given a tuple $\langle t, e, s\rangle$, and a target transition $(t, e) \xrightarrow{a} (t', e')$, $MATCH(s, e \xrightarrow{a} e', t \xrightarrow{a} t')$ tries to find a transition of the community such that property (P1) above is true. It is possible that there could be multiple services that can make an a-transitions from the current state of the community. The $MATCH$ function needs to try them one by one until it finds a match. To this end, $MATCH$ maintains all potentially valid community transitions in a queue.

QUEUE. The algorithm maintains a queue that holds all potential transitions of the community from state (s, e) that can potentially match a target transition $(t, e) \xrightarrow{a} (t', e')$. Once it finds that a transition does not match, it discards it and dequeue the next possible transition. The algorithm keeps doing this until a matching transition is found or the queue becomes empty. If the queue becomes empty then there is no match and the function $MATCH$ returns false: no matching service is found.

The use of a queue is not strictly necessary since one can iterate over all the possibilities one by one. It is included because we envision the use of a priority queue where the priority is assigned for a given service according to some user preference to implement non-functional requirements, or as a quality of service weight. Also, if the algorithm uses a heuristic based on some already obtained information that makes one transition more likely to succeed, it will be given higher weight. Finally, the queue is defined in such a way that if there are, in a given service k, many s'_k such that $s_k \xrightarrow{a} s'_k$ then $ENQUEUE(Q, k, s')$ will add the first such s'_k only. This is because from the definition of the controllability relation if $(t, e) \xrightarrow{a} (t', e')$ then it is enough to find a **single** matching transition. The reverse, namely property (P2) is checked on lines 12-15.

3.3.2 Example 2

Even with the simplifications of using the special case of the Roman Model the algorithm is still a little bit complicated to follow. To give an intuitive idea to the reader how the algorithm works we use a very simple scenario without an environment shown in Figure 3.4 and go through the algorithm step by step with the call stack shown in Figure 3.5 as a guide.

The first call is to the starting states (s_0, t_0) by calling $CONTROL(s_0, t_0)$. Tuple (s_0, t_0) is added to $\mathcal{A}$ (line 1 in the call stack in Figure 3.5). Then every possible transition of t_0 is matched. In the present example there is a single transition $t_0 \xrightarrow{a} t_1$ which is matched on line 2. Once the function MATCH is called there are two possibilities: $s_0 \xrightarrow{1a|a} s_1$ and $s_0 \xrightarrow{2a|a} s_2$. Each of these possibilities is tried in turn until one of them (or none) matches. The choice of the order is unspecified. In this example, for illustration purposes, we choose the transition $s_0 \xrightarrow{1a|a} s_1$ first (line 3). As can be seen from lines 4-10 the call to $CONTROL(s_1, t_1)$ returns false. Therefore the second possibility is tried (line 11) by calling $CONTROL(s_2, t_1)$ which returns true and therefore $CONTROL(s_0, t_0)$ returns true which means the community is controllable with respect to the target and hence an orchestrator exists and can be synthesized from the controllability relation

Algorithm 3: Main routine for computing the controllability relation

```
1 MAIN
2 B ← ∅
3 while changed=true do
4 |   A ← ∅
5 |   changed=false
6 |   CONTROL(t^0, e^0, s^0)
7 return A
```

A.

The fact that the states are visited in a preorder traversal but processed in a postorder traversal causes a problem that needs to be handled. This happens because once a pair (s, t) is removed from $\mathcal{A}$, the algorithm needs to check for other states that depend on (s, t) as "proof" that these states are in the controllability relation. We illustrate using the example shown in Figure 3.6. In that example there is no orchestrator for the target and yet if we run the algorithm it will tell us otherwise. Stepping through the algorithm as shown in Figure 3.7 and Figure 3.8 will identify the cause of the problem.

Recall that in the algorithm nodes are added to $\mathcal{A}$ in a preorder traversal. Therefore initially (s_0, t_0) is assumed to be in the controllability relation and added to $\mathcal{A}$ (line 1). Then the matching process begin where one tries to find a match to all the transitions of t_0. Since t_0 has only an single transition $t_0 \xrightarrow{a} t_1$ then $MATCH(s_0, t_0 \xrightarrow{a} t_1)$ is called (on line 2). The two possible a transitions of s_0 are tried in turn. The first one fails since the call to $CONTROL(s_1, t_1)$ returns false as it should, but has a **side effect** of adding (s_2, t_2) to $\mathcal{A}$ (see line 22 in Figure 3.7). The reason this happened is that the call to $CONTROL(s_2, t_2)$ (line 16 Figure 3.7) returns **before** (s_1, t_1) was removed from $\mathcal{A}$ on line 22 .

At this point $\mathcal{A} = \{(s_0, t_0), (s_2, t_2)\}$ and the call to $CONTROL(s_4, t_1)$ as seen in Figure 3.8 returns true. One can see by inspecting line 5 on Figure 3.8 that one of "proofs" that (s_4, t_1) is in the controllability relation is that there is a b-transition to a tuple in the controllability relation, in this case (s_2, t_2) which is wrong.

For this reason the algorithm maintains a variable *changed* that is set to true every time a tuple is removed from the set A. If after the algorithm finishes $changed = true$ then there is a possibility that the aforementioned case occurred and the algorithm should be run again:
Actually not every time a change occurs one needs to rerun the algorithms. Sometimes the removed tuple does not affect the rest of the computation. But if we don't rerun the algorithm every time that $change = true$ then one needs to keep track of the dependencies of each state and rerun the algorithm on the dependencies which makes the algorithm much more complicated than it already is. Therefore we opt for the simple process of rerunning the algorithms without checking for dependencies. Note that this approach could make *at most* one redundant call to the algorithm. Finally it should be mentioned that the presented algorithm computes one of

the controllability relations between the target and the community if one exists. As we have mentioned in chapter 2 one can compute the orchestrator generator by keeping track of the transitions. In that case, the relation $\mathcal{A}$ becomes a graph where each node is a tuple (t, s) as before and in addition the graph edges represent the transition "proofs".

Note that at the start of every run, or pass, the relation $\mathcal{A}$ is destroyed (i.e. set to $\emptyset$) whereas the relation $\mathcal{B}$ that keeps track of all tuples that are not in the controllability relation is carried from one pass to the other. As will be shown in the next section it is guaranteed that *changed* will eventually be false and the algorithm will terminate.

3.4 Correctness and complexity of the algorithm

Let n be the number of available services, with each service having N_i states, N_t the number of target service states, and N_e the number of environment states. Let $N = N_t \times N_e \times N_1 \times \ldots \times N_n$.

Theorem 3.4.1. *The algorithm CONTROL terminates in a finite number of steps and when it does it returns true iff (t^0, e^0, s^0) is the controllability relation.*

Proof. First we prove the termination. Let $CONTROL_i$ be the i^{th} iteration of $CONTROL(t^0, e^0, s^0)$ and $\mathcal{B}_i$ the set of tuples that are not in the controllability relation after $CONTROL_i$ finishes. The variable *changed* is set to *true* iff during the run $\exists (t, e, s) \notin \mathcal{B}_{i-1}$ and $(t, e, s) \in \mathcal{B}_i$, meaning that (t, e, s) was found to be not in the controllability relation during the execution of $CONTROL_i$. Recall that at no point in the algorithm, tuples are removed from $\mathcal{B}$. But if no new state is added to $\mathcal{B}$ then the algorithm stops. Therefore the set $\mathcal{B}$ is strictly increasing. On the other hand, the total number of tuples N is finite. Then there is an iteration j such that the variable $change = false$ and at that point the algorithm terminates.

Next we show that it yields the correct result. Observe that in a given iteration i of the algorithm, we have that if $CONTROL(t, e, s)$ returns *true* it means that it has finished processing the state $\langle t, e, s \rangle$ and that $\langle t, e, s \rangle \in \mathcal{A}$. Also, recall that it returns *true* iff for every a:

1. And for every transition $t \xrightarrow{a} t'$ and for all $e \xrightarrow{a} e'$ there exists a community transition $s \xrightarrow{ka|a} s'$ such that $\langle t', e', s' \rangle \in \mathcal{A}$.

2. And for every s'' such that $s \xrightarrow{ka|a} s''$ it is the case that $\langle t, e, s'' \rangle \in \mathcal{A}$.

The above two conditions hold in every iteration when $CONTROL$ returns. In particular, in the final iteration, when $changed = false$ and therefore no $\langle t', e', s' \rangle$ was removed from $\mathcal{A}$, imply that the above two conditions hold for all $(t, e, s) \in \mathcal{A}$ and therefore relation $\mathcal{A}$ is a controllability relation.

□

Theorem 3.4.2. *The algorithm CONTROL is polynomial in the number of states of a given service and exponential in the number of services.*

First recall that $N = N_t \times N_e \times N_1 \times N_2 \times \ldots \times N_n$ is the number of possible states of the community and target combined. Since we are doing a worst-case analysis, we assume that all the above states are reachable.
We begin the complexity analysis by considering a **single run** of $CONTROL(t^0, e^0, s^0)$. In one such run each state is considered once. This is because after the first visit it is either in $\mathcal{A}$ or in $\mathcal{B}$. On any subsequent call it will not be visited again (lines 2-5 in $CONTROL$). This means that each call of $CONTROL(t^0, e^0, s^0)$ considers at most N states. Next we compute the cost of visiting a single state. The loops in lines 8-12 have the following cost:

$$\begin{aligned}&\sum_a |\{\delta(t,a) \times \delta_E(e,a)\}| \cdot |MATCH(s, e \xrightarrow{a} e', t \xrightarrow{a} t')| \\ &\leq \sum_a |\{\delta(t,a) \times \delta_E(e,a)\}| \cdot |\{\delta(s,a)\}|\end{aligned}$$

The last equality is true because for a given $t \xrightarrow{a} t'$ and $e \xrightarrow{a} e'$ the function $MATCH$ will process at most $|\{\delta(s,a)\}|$ transitions. Recall that one needs to match a community s state with a target state t for every environment state e. In other words one cannot match a community state s in environment state e_1 with a target state t with a **different** community state e_2. This is why the $\delta_E(e,a)$ in the above is counted once, in the term related to the number of target states.

The above is the contribution of a single state. Because every state is visited at most once the total cost of one iteration of $CONTROL$ is

$$\begin{aligned}&\leq \sum_e \sum_t \sum_s \sum_a |\{\delta(t,a) \times \delta_E(e,a)\}| \cdot |\{\delta(s,a)\}| \\ &\leq \left(\sum_a \sum_t |\{\delta(t,a)\}|\right) \cdot \left(\sum_a \sum_e |\{\delta_E(e,a)\}|\right) \cdot \left(\sum_a \sum_s |\{(\delta(s,a)\}|\right) \\ &= |L_t| \cdot |L_e| \cdot |L_s| \end{aligned} \tag{3.7}$$

Where $|L_t|$, $|L_e|$, and $|L_s|$ are the number of transitions of the target, environment and the community respectively.

To get an idea about the complexity of the algorithm as a function of the number of services, n, we note that for a given action a, if service i can make $|L_{ai}|$ transitions then the asynchronous product can make $\prod_i |L_{ia}|$. Therefore:

$$|L_s| = \sum_a |L_{1a}| \cdots |L_{an}|$$

In the worst-case every state has an a-transition to every other state. Thus $|L_{ia}|= O(N_i^2)$, where N_i is the number of states in service i. Using the same analysis for the target and environment transitions we get that the complexity of processing a *single* iteration of $CONTROL$ is

$$O(N_t^2 \cdot N_e^2 \cdot N_1^2 \cdots N_n^2)$$

Since $CONTROL$ is called at most $O(N_t \cdot N_e \cdot N_1 \cdots N_n)$ times on (t^0, e^0, s^0) ($s^0 = \langle s_1^0, \ldots, s_n^0 \rangle$), the total complexity is $O(N_t^3 \cdot N_e^3 \cdot N_1^3 \cdots N_n^3)$. Therefore, the algorithm is polynomial in the number of states of target, environment, or a given services. It is exponential in the number of services. Considering that the problem is EXPTIME-hard [MW08], this is optimal.

3.5 Handling service failure

In complex systems it is possible for some of the available services to fail. This could be due to the failure of the service itself, its underlying platform or the communication channel. One advantage of having the *largest* solution, i.e. one that contains all possible solution, is that when some services fail, one can switch to a solution that does not used the failed service. In this section we argue that this is show that the proposed on-the-fly algorithm is also robust to service failure.

Let $s = (s_1, \ldots, s_n)$ and suppose that the community is in state (t, e, s) and service S_k fails for some reason so the orchestrator cannot delegate actions to it. If the target makes a transition $t \xrightarrow{a} t'$, then the *largest* solution, since it contains all possible solutions, can choose (if one exists) an alternative service l, to make a $s \xrightarrow{la|a} s''$ transition instead of $s \xrightarrow{ka|a} s'$. If the service failure is of a very short duration, i.e. the service $\mathcal{S}_k$ becomes available again right after the transition, this decision is done at no cost. In such a situation finding the *largest* solution, which contains all possible solutions, has an advantage over the our proposed method.

But if the service failure is permanent or of a long duration the tuple (t', e', s'') is not necessarily in the controllability relation anymore and any method has to recompute the solution again. It should be noted that term "long duration" means the time it takes to make one extra transition which is not long at all. While failure time cannot be quantified it is reasonable to say that the vast majority of failures are classified as "long duration". In [GPS13] they show, that when using a fixpoint algorithm, one does not need to recompute the controllability relation from scratch, but can use information obtained before the failure.

In this section we show that the recomputation of the solution using the proposed algorithm also does not have to start from scratch, but can make use of the results of the computation before a service failure, to speed up the computation after a service failure. In particular the set $\mathcal{B}$, used in the algorithm, is kept because every state in $\mathcal{B}$ before failure will also be in $\mathcal{B}$ after failure. This way the algorithm does not have to revisit, and process, those states again, which results, depending on the size of $\mathcal{B}$, in significant pruning of the state space.

The main idea centers around the following theorem.

Theorem 3.5.1. *After the algorithm finds a solution if* $(t, e, s) \in \mathcal{B}$ *and service* l *fails then* $(t, e, s) \in \mathcal{B}$ *after failure.*

Proof. Let $s = (s_1, \ldots, s_n)$ and R be the largest controllability relation. From definition (3.6) we have that $(t, e, s) \notin R$ implies that $(t, e, s) \notin R_{i+1} \wedge (t, e, s) \in R_i$ for some i. Now $(t, e, s) \notin R_{i+1}$ implies that

$$\exists a, t', e'.t \xrightarrow{a} t' \wedge e \xrightarrow{a} e' \wedge \Big(\delta(s, e, a) = \emptyset \vee \forall k(\exists s'.s \xrightarrow{ka|a} s' \wedge (t', e', s') \notin R_i)\Big) \tag{3.8}$$

The above clause is basically saying that: there is a target transition that cannot be matched by a community transition. This mismatch can happen in two ways: either the community cannot make an a-transition ($\delta(s, e, a) = \emptyset$) or the community can make the transition but one of them will lead to a non controllable states which means eventually the community cannot model the target.

One can break the clause in (3.8) in two parts, one that applies for $i = 0$ and the other for $i > 0$. Namely:

1. $i = 0$ then $(t, e, s) \notin R_1$ implies $\exists a, t', e'.t \xrightarrow{a} t' \wedge e \xrightarrow{a} e' \wedge \delta(s, e, a) = \emptyset$.
2. $i > 0$ then $(t, e, s) \notin R_{i+1}$ implies

$$\exists a, t', e'.t \xrightarrow{a} t' \wedge e \xrightarrow{a} e' \wedge \forall k \exists s''.s \xrightarrow{ka|a} s'' \wedge (t', e', s'') \notin R_i$$

Remark 3.5.2. It is worth mentioning that the property in case (2) is different from the property: $\forall s' \in \delta(s, e, a)$ we have $(t', e', s') \notin R_i$, which means that **all** the a-transitions will lead to tuples not in R. What the definition is actually saying is that, for a given (t, e, s), there exists an action a that can be performed by the target and environment, but no matter which service one selects to delegate the action to, it will lead to *at least* one tuple (t', e', s') which is not in R. This happens because one does not have total control over the services due to non-determinism. This uncontrollability is more accentuated in the general model where, in addition to the non-determinism of individual services, there is an additional uncontrollability because when a message α is sent it could force the community to make some b-transitions in addition to the desired a-transition

We show by induction on the number of iterations i, that when a service fails, if $(t, e, p) \notin R$ before failure then it remains so after the failure.

Base case: This case follows directly from case (1) above and it is true for all $(t, e, s) \notin R_1$. The fact that $(t, e, s) \notin R_1$ means that there exists a transition $t \xrightarrow{a} t' \wedge e \xrightarrow{a} e'$ which cannot be matched by any service k, $\delta(s, e, a) = \emptyset$. Since the community is an asynchronous product of all services, which means that $\delta(s, e, a)$ is the union of the a-transitions of all services then clearly when a certain service is no more available we would still have $\delta(s, e, a) = \emptyset$. Therefore if $(t, e, p) \notin R_1$ before failures it remains $\notin R_1$ after failure.

Hypothesis: Assume that the property is true for iteration i: if $(t', e', s') \notin R_i$ before failure then $(t', e', s') \notin R_i$ after failure.
Induction step: Consider $(t, e, s) \notin R_{i+1}$ before failure. From the definition of not in R_{i+1} we have:

$$\exists a, t', e'.t \xrightarrow{a} t' \wedge e \xrightarrow{a} e' \wedge \forall k(\exists s''.s \xrightarrow{ka|a} s'' \wedge (t', e', s'') \notin R_i)$$

Suppose that service m fails then the above becomes:

$$\exists a, t', e'.t \xrightarrow{a} t' \wedge e \xrightarrow{a} e' \wedge \forall k, k \neq m(\exists s''.s \xrightarrow{ka|a} s'' \wedge (t', e', s'') \notin R_i)$$

On the other hand, by the induction hypothesis all $(t', e', s'') \notin R_i$ after failure and thus $(t, e, s) \notin R_{i+1}$ after failure. □

As the above theorem shows, the algorithm we are proposing is robust in the case of failure. If a service fails after the controllability relation (and consequently the orchestrator) is computed, then computing the controllability relation again reuses information obtained before the failure.

3.6 Abstraction of the composition problem

One of the important advantages of the proposed on-the-fly algorithm over so called "fixpoint algorithms" is the ability to combine it with some heuristics which will allow it to prune the search space and reduce its cost. Toward that end and as a proof of concept we develop in this section an abstraction method, the results of which can be used by the proposed algorithm as a heuristic. In fact the method does more than that. It also allows us to infer the non-existence of an orchestrator for the problem under study from the non-existence of a particular relation for a much smaller, abstracted community. While the subject of abstraction has been studied extensively in the context of model checking[CGL94][CGJ$^+$03] to our knowledge it has not been applied to the case of service composition.

The main concern of this part is an abstraction technique that reduces the number of states of the composition problem and therefore make the orchestrator synthesis more efficient. Basically, an abstraction is a function that reduces the size of the original problem but preserves some of the properties under study. When done properly, the abstraction allows one to infer properties of the original community by studying the, much reduced, abstract community.

The abstraction proposed in this part allows one to infer the non-existence of an orchestrator for the original community and target from the non-existence of a simulation relation between the, much smaller, abstracted community and the abstracted target. More importantly, when a simulation *does exist* between the abstracted community and the target, the result is used as a *branch-and-bound like* heuristic for the on-the-fly algorithm to speed up the computation of the controllability relation, and therefore the orchestrator, between the original community and target. The abstraction method is presented in section 3.6.1 and its integration in the algorithm is discussed in section 3.6.2.

3.6.1 Quotient services and state reduction

The sought abstraction reduces the number of states of the community by using the concept of the *quotient* of a Labelled Transition System(LTS). Given an equivalence relation α on the states of an LTS, the idea is to group all equivalent states into a single "super" state.

Definition 3.6.1. *Let $\mathcal{S} = \langle S, \Sigma, s^0, \delta \rangle$ be an LTS and α an equivalence relation on the states of $\mathcal{S}$. An* equivalence class *of α, denoted by $[p]_\alpha$, is the set $[p]_\alpha = \{q \in S \mid (p,q) \in \alpha\}$ and the* quotient of $\mathcal{S}$ by α *is an LTS defined as, $\mathcal{S}/\alpha = \langle [S]_\alpha, \Sigma_\alpha, [s^0], \delta_\alpha \rangle$ where*

- $\Sigma_\alpha \subseteq \Sigma$ *is a subset of the original action set, which will be fixed later.*
- $[S]_\alpha = \{[s] \mid s \in S\}$ *is the set of equivalence classes of α.*
- $[s^0] = \{s \in S \mid (s, s^0) \in \alpha\}$ *is the equivalence class of the initial state.*
- $\delta_\alpha \subseteq [S]_\alpha \times \Sigma_\alpha \times [S]_\alpha$ *is the transition relation of the quotient defined as:*

$$([p], a, [q]) \in \delta_\alpha \text{ iff } \exists x \in [p], y \in [q].(x, a, y) \in \delta$$

One important equivalence relation that can be used is bisimulation equivalence. Choosing bisimulation equivalence has the distinct advantage of retaining **all** the properties of the original system that one might care to study. However, state reduction by using bisimulation equivalence is minimal [WHH$^+$06] and a much coarser relation is need.

We don't do a systematic study of different equivalence relation but rather show one equivalence (actually a pair of) relation that works and at the same time offer a substantial reduction in state space. According to the results in [WHH$^+$06] branching bisimulation offers up to 4 orders of magnitude of reduction in state space. Also one would expect even more impressive result from the coarser (and therefore less restrictive) *closure relation* that will be introduced later.

To be able to reduce the systems under study the action alphabet Σ is divided into two parts: $\Sigma = \Sigma_a \uplus \Sigma_c$ where transitions in Σ_a are to be "abstracted away" and will become "unobservable" in the abstracted system. For the target service $\mathcal{S}_t = \langle S_t, \Sigma, t^0, \delta_t \rangle$, define the relation $\longrightarrow_{\Sigma_a} = \{(t, t') \in S_t \times S_t \mid \exists a \in \Sigma_a \text{ with } t \xrightarrow{a} t'\}$. For the community $\mathcal{S} = \langle S, \Sigma, \{n\}, s^0, \delta_u \rangle$ the same notation is used for a similar relation $\longrightarrow_{\Sigma_a} = \{(s, s') \in S \times S \mid \exists k \in \{n\}, a \in \Sigma_a \text{ with } s \xrightarrow{ka|a} s'\}$. Let $\longrightarrow^*_{\Sigma_a}$ be the reflexive, transitive closure of $\longrightarrow_{\Sigma_a}$. It will be inferred from context if $\longrightarrow^*_{\Sigma_a}$ refers to the target or the composition.

The target is abstracted using the largest branching bisimulation [vGW96].

Definition 3.6.2 (Branching bisimulation)**.** *Given a target LTS $\mathcal{S}_t = \langle S_t, \Sigma, t^0, \delta_t \rangle$ with $\Sigma = \Sigma_a \uplus \Sigma_c$, a relation $R \subseteq S_t \times S_t$ is called a branching bisimulation on $\mathcal{S}_t$ if it is symmetric and $\forall (p,q) \in R$ and $\forall \alpha \in \Sigma$ if $p \xrightarrow{\alpha} p'$ then*

- *Either $\alpha \in \Sigma_a$ and $(p', q) \in R$*

- *Or* $\exists q', q''$ *such that* $q \longrightarrow^*_{\Sigma_a} q' \xrightarrow{\alpha} q''$ *with* $(p, q') \in R$ *and* $(p', q'') \in R$.

The quotient of the target by the branching bisimulation is denoted by $\bar{\mathcal{S}}_t = \langle [S_t], \Sigma_c, [t^0], \delta_b \rangle$, where we have dropped the subscript b from $[S_t]$ since only branching bisimulation will be used to abstract the target. The transition relation δ_b is the transition relation between equivalence classes of the branching bisimulation as given in definition 3.6.1.

The community is abstracted using the *closure relation*.

Definition 3.6.3 (Closure relation)**.** *Let* $\mathcal{S} = \langle S, \Sigma, \{n\}, s^0, \delta_u \rangle$ *be the community of* n *available services. We define the closure relation,* $\mathcal{C}$*, as the reflexive, transitive closure of the relation* $\rho = \{(s, s') \in S \times S \mid \exists a \in \Sigma_a, k \in \{n\}, s \xrightarrow{ka|a} s' \text{ or } s' \xrightarrow{ka|a} s\}$.

The quotient of the community by the closure relation is $\bar{\mathcal{S}} = \langle [S], \Sigma_c, \{n\}, [s^0], \delta_c \rangle$ where δ_c is the transition between the equivalence classes and we have dropped the subscript from $[S]$ since only the closure relation is used to abstract the community.

Next we relate the abstracted target to the abstracted community using a *variant* of the classical simulation relation [Mil71]. In fact, if there is a simulation relation, as defined below, between the abstracted target and the abstracted community it means that there is an orchestrator such that the *orchestrated* abstracted community simulates, in the classical sense, the abstracted target.

Definition 3.6.4 (Simulation)**.** *Given an abstracted target service* $\bar{S}_t = \langle [S_t], \Sigma_c, [t^0], \delta_b \rangle$ *and an abstracted community service* $\bar{S} = \langle [S], \Sigma_c, \{n\}, [s^0], \delta_c \rangle$ *a relation* $\rho \subseteq [S_t] \times [S]$ *is said to be a simulation iff for all* $(t, s) \in \rho$ *and all* $c \in \Sigma_c$ *we have*

- $t \xrightarrow{c} t' \Rightarrow \exists k \in \{n\}. s \xrightarrow{kc|c} s'$ *and* $(t', s') \in \rho$.

We say that $\bar{S}$ *simulates* $\bar{S}_t$*, and we write* $\bar{S}_t \prec \bar{S}$ *iff there exists a simulation* ρ *such that* $([t^0], [s^0]) \in \rho$.

Example 3.6.1. *An example community and target are shown in Figure 3.9. The above abstractions are used on the example with the "buy" transitions abstracted away, i.e.* $\Sigma_a = \{buy\}$ *and* $\Sigma_c = \{search, pay, clear\}$*. The result of the abstraction is shown in Figure 3.10. In part (A) the dashed ovals denote the equivalence classes of the abstracted community. Also the actions search, buy, pay and clear are shortened in part (A) to s, b, p and c respectively and the actions, which can be inferred from the messages, are omitted for clarity.*

The next theorem gives the main result of this section.

Theorem 3.6.2. *Let* $\mathcal{S}_t$ *and* $\bar{S}_t$*,* $\mathcal{S}$ *and* $\bar{S}$ *be the target and community LTS together with their respective abstractions. If* $\mathcal{S}$ *is controllable with respect to* $\mathcal{S}_t$ *then* $\bar{S}_t \prec \bar{S}$.

Proof. Let R be the controllability relation between $\mathcal{S}_t$ and $\mathcal{S}$. Define the relation $R_{[]}$ between $\bar{\mathcal{S}}_t$ and $\bar{\mathcal{S}}$ with $R_{[]} = \{([x],[y]) \mid (x,y) \in R\}$. We prove that $R_{[]}$ is a simulation relation (see Figure 3.11).
Let $([x],[y]) \in R_{[]}$ and $[x] \xrightarrow{a} [u]$. From the definition of the quotient LTS we get that $\exists x_1 \in [x], u_1 \in [u]$ such that $x_1 \xrightarrow{a} u_1$. Since $(x,x_1) \in \alpha$ and α is a branching bisimulation then $\exists x_2 \in [x], u_2 \in [u]$ with $x \longrightarrow^*_{\Sigma_a} x_2 \xrightarrow{a} u_2$.
On the other hand $(x,y) \in R$ therefore $\exists y_2, v_1, k$ such that $y \longrightarrow^*_{\Sigma_a} y_2 \xrightarrow{ka|a} v_1$ with $y_2 \in [y]$, $(x_2,y_2) \in R$ and $(u_2,v_1) \in R$. This implies that $\exists [v]$ such that $[y] \xrightarrow{ka|a} [v]$ and $([u],[v]) \in R_{[]}$. Therefore $R_{[]}$ is a simulation relation. □

It is worth mentioning that in the example shown in Figure 3.10that the abstracted community is *not* controllable with respect to the abstracted target. For example, given the abstracted pair (S_1,T_1) when $T_1 \xrightarrow{pay} T_2$, the abstracted community has two possible *pay* transitions: one to S_2 with $(T_2,S_2) \in R_{[]}$, the second is S_0 with $(T_2,S_0) \notin R_{[]}$.

We are actually interested in the case when no simulation exists between the abstracted target and abstracted community. The theorem below is obtained by combining theorem 2.3.4 which links the existence of an orchestrator to the existence of a controllability relation and theorem 3.6.2.

Theorem 3.6.3. *If no simulation relation exists between the abstracted target and the abstracted community then no orchestrator Ω exists, such that the orchestrated community is a composition of the target.*

It should be emphasized that the abstracted community $\bar{S}$ is independent of the target service and therefore it is computed once, offline, and will be used for any target service. The choice of which actions to abstract depends on the situation. For example, one can abstract away non essential action. Another approach would be to abstract away the action that occur most and thus obtain maximum reduction in transitions. Obviously the abstraction of the goal has to be done for every request goal and which cannot be precomputed. But considering that the intractability of the behavior composition problem comes from the available services and not from the target this method is extremely promising.

There are two possible outcomes for the computation of a simulation between the abstracted LTSs:

1. No simulation exists. Considering that the closure relation is coarser than branching bisimulation which can reduce the original system by up to 4 orders of magnitude [WHH+06] , this result guarantees, in much less computational steps, that no composition exists for the original system.

2. A simulation exists. This does not guarantee the existence of an orchestrator for the original system. But one can use this result as follows: the simulation that was computed for the

Algorithm 4: function MATCH when abstraction is used

```
1  MATCH(s, e --a--> e', t --a--> t')
     /* s_i is the i^th component of s,i.e.  s = ⟨s_1, ..., s_n⟩.  Similarly for s' and s''
        */
2  for i = 1 to n do
3      foreach s_i --g(e),a--> s'_i do
           /* Determine the equivalence classes                                  */
4          x = D[t']
5          y = C[s']
           /* Consider a transition ONLY if the equivalence classes are similar
              */
6          if R[x][y] = 1 then
7              ENQUEUE (Q,i,s'_i)
8  res = false
9  while Q ≠ ∅ ∧ res = false do
10     s'_k = DEQUEUE(Q)
11     res=CONTROL(t', e', s')
12     if res=false then
13         Goto Label
14     else
15         foreach s_k --a--> s''_k do
16             res=CONTROL(t', e', s'')
17             if res = false then
18                 Goto Label
19     Label:
20 return res
```

abstracted LTSs is used as a guide for a heuristic to speed up the computation of the controllability relation for the original problem as will be shown in the next section.

3.6.2 Heuristic for orchestrator synthesis

In this section we present a heuristic based on the abstraction result. To do so it is assumed that $\bar{S}_t \prec \bar{S}$ and the result can be used as input to the on-the-fly algorithm in order to speed up its execution. Now suppose that $\bar{S}_t \prec \bar{S}$ and that the simulation relation $R_{[]}$ between the states of $\bar{S}$ and $\bar{S}_t$, was already computed. The next step is to incorporate the information obtained from $R_{[]}$ in the algorithm presented in Section 3.3. This is done by discarding some transitions that are not compatible with the information obtained from $R_{[]}$ which will save the algorithm a lot of

backtracking.
Given a pair of target and community states $\langle t, s\rangle$ and an arbitrary target transition $t \xrightarrow{a} t'$, the algorithm needs to find a *matching* community transition $s \xrightarrow{ka|a} s'$. This means that $\langle t', s'\rangle$ should belong to the controllability relation. On the other hand, the result of theorem 3.6.2 implies that we should have $([t'], [s']) \in R_{[]}$. In other words, among the many possible transitions $s \xrightarrow{ka|a} s''$ the algorithm should try only the ones such that $([t'], [s'']) \in R_{[]}$. This simple yet powerful idea is incorporated in the $MATCH$ function with the result shown in Algorithm 4.
To illustrate the heuristic we consider Figure 3.10 again. Note that all similarly labeled transitions from the same source class to the same destination class are combined. The relation $R_{[]}$ is a simulation between the abstracted target and abstracted community.

One can see how the heuristic improves the composition problem by considering the starting states t_0 and (u_0, v_0). From the initial target state, the target can make a *search* transition $t_0 \xrightarrow{search} t_1$. In the abstracted target it is a transition from T_0 to T_1. The community has two possible *search* transitions: $(u_0, v_0) \xrightarrow{1search} (u_1, v_0)$ and $(u_0, v_0) \xrightarrow{2search} (u_0, v_1)$. From Figure 3.10 one can see that the first transition leads to class S_0 and the second one to class S_1. Since $(S_1, T_1) \in R_{[]}$ then it is considered as a potential transition whereas $(S_0, T_1) \notin R_{[]}$ therefore it is discarded because according to theorem 3.6.2 it cannot lead to a solution. Since the discarded choice can represented a significant portion of the search space, this branch-and-bound like heuristic can save a lot of search time.
To implement these rules we define three data structures. An associative array C, indexed by the states of the community, stores the equivalence classes of $\bar{\mathcal{S}}$. Then, $C[s] = k$ means that the state s belongs to the equivalence class number k. Note that the array C depends only on the available services and therefore it is *precomputed* offline and it is *the same* for all target services. Similarly, an associative array D is used for the target such that $D[t] = l$ means that target state t is in equivalence class l. Array D is dependent on the goal service only. Finally, the relation $R_{[]}$ is represented as a two dimensional array where $R[D[t]][C[s]] = v$, $(v \in \{0, 1\})$, means that the equivalence classes $D[t]$ and $C[s]$, of the target and community, respectively, are similar when $v = 1$ and not similar when $v = 0$. The relation R is computed for the abstracted system. If $R[D[t_0]][C[s^0]] = 0$ then we know that there is no composition for the system. Otherwise, R is used in the local search algorithm as shown in the $MATCH$ function in Algorithm 4 on lines 4-6 . Note the minimal overhead incurred since an array access is $O(1)$

3.7 Algorithm for the general model

For simplicity of the presentation we have so far restricted ourselves to the special case of Roman Model. In this section we present the extension of the on-the-fly algorithm given in Section 3.3 for the Roman Model to our more general model. Recall from Chapter 2 that for our more general model the orchestrator (with perfect information) does not have full controllability over the community. In other words, there are some community transitions that happen regardless

Algorithm 5: function CONTROL for the general model

```
1  CONTROL(t, e, s)
2  if ⟨t, e, s⟩ ∈ B then
3  |  return false
4  if ⟨t, e, s⟩ ∈ A then
5  |  return true
   /* Assume that (t,e,s) are in controllability relation                    */
6  A = A ∪ ⟨t, e, s⟩
7  res = true
8  Θ ← ∅
   /* Check that all uncontrolled community transitions can be matched by a
      May transition of the target                                           */
9  foreach a ∈ Σ do
10 |  foreach s --g(e),a--> s' ∧ e --a--> e' do
11 |  |  res = ∃t'.t --g(e),a--> t' ∧ CONTROL(t', e', s')
12 |  |  if res = false then
13 |  |  |  Goto Exit
14 |  |  Θ ← Θ ∪ {a}
   /* Check if orchestrator can send messages to match the remaining Must
      transitions of the target                                              */
15 foreach a ∈ Σ − Θ do
16 |  foreach t --g(e),a--> t' ∧ e --a--> e' do
17 |  |  res=MATCH (s, e --a--> e', t --a--> t')
18 |  |  if res = false then
19 |  |  |  Goto Exit
20 Exit:
21 if res=false then
22 |  B = B ∪ ⟨t, e, s⟩
23 |  A = A − ⟨t, e, s⟩
24 |  changed = true
25 return res
```

of the action of the orchestrator. The remaining transition can be controlled by the orchestrator by sending messages from a set Com. Unlike the Roman Model where a message ia can control which service, in this case i, and which action to enable, in this case a, in our general model a message α can enable different services and different actions at the same time. Also, we will use modal specification for the target service. Recall that in modal specification the target service is a tuple $\mathcal{S}_t = \langle S_t, \Sigma, t^0, Must, May, G\rangle$ where $May \subseteq S_t \times G \times \Sigma \times S_t$ is a set of transitions

Algorithm 6: function MATCH for the general model

```
1 MATCH((s, e -a-> e', t -a-> t')
2 foreach α ∈ Com do
3     if s -g(e),α|a-> s' ∧ CONTROL(t', e', s') then
          /* Found a match by using α.  Now check it doesn't cause "side
             effects"                                                    */
4         foreach s -g(e),α|b-> s'' ∧ e -b-> e'' do
5             res = ∃t''.t -g(e),b-> t'' ∧ CONTROL(t'', e'', s'')
6             if res = false then /*this α does not work, try another one*/
7                 break;
8         return true
  /* No α matched                                                       */
9 return false
```

(transition relation) that the community *can*, but is not forced to, match and $Must \subseteq May$ is a set of transitions that the community *must* match. Note that we have added the effects of the environment, through the set of constraints functions G, in this section. We use dashed and solid arrows for the *May* and *Must* transitions respectively. If the environment is in state e and the target is in state t we write $t \xrightarrow{g(e),a} t'$ for $\exists g \in G.(t, g(e), a, t') \in May \wedge g(e) = true$ and $t \xrightarrow{g(e),a} t'$ for $\exists g \in G.(t, g(e), a, t') \in Must \wedge g(e) = true$. From a fixpoint perspective the equivalent of (3.4) for the Roman Model is the following function over the set of relations:

$$F(R) = \Big\{ (t,e,s) \in R \mid \forall a \\ \Big[t \xrightarrow{g(e),a} t' \wedge \delta_E(e,a) \neq \emptyset \Rightarrow \mathcal{C} \Big] \\ \wedge \\ \Big[s \xrightarrow{g(e),a} s' \wedge \delta_E(e,a) \neq \emptyset \Rightarrow \exists t'.t \xrightarrow{g(e),a} t' \wedge \forall e' \in \delta_E(e,a), (t',e',s') \in R \Big] \Big\}$$

where $\mathcal{C}$ is the expression defined by

$$\exists s'.s \xrightarrow{g(e),a} s' \wedge \forall e' \in \delta_E(e,a), (t',e',s') \in R \\ \vee \Big[\exists \alpha, s'.s \xrightarrow{g(e),\alpha|a} s' \\ \wedge \forall b (s \xrightarrow{g(e),\alpha|b} s'' \wedge \delta_E(e,b) \neq \emptyset \Rightarrow \exists t''.t \xrightarrow{g(e),b} t'' \wedge \forall e'' \in \delta_E(e,b), (t'',e'',s'') \in R) \Big]$$

Using the function F defined above, a solution to the problem (i.e. finding a controllability relation, R, between the target and the community) is found by iterating the function F, starting from the full state space, until a fixpoint is reached. Formally,

$$R_0 = S_t \times E \times S$$
$$R_{i+1} = F(R_i)$$

As discussed in previous sections since $S_t \times E \times S$ is finite then $\exists j.F(R_j) = R_j$ and R_j is the sought controllability relation.

Similarly to what we have done in Section 3.3, we developed an on-the-fly algorithm that uses two mutually recursive functions $CONTROL$ and $MATCH$ which are shown in Algorithms 5 and 6.

Given a state (t, e, s), algorithm 5 returns true only if (t, e, s) is in the *controllability* relation. The function $CONTROL$, first checks if the *uncontrollable* community transitions from state s can be matched by a *May* target transitions (lines 9-14). Unlike the controllable transitions, if a **single uncontrollable** transition cannot be matched by a *May* transition of the target (lines 11-13) then (t, e, s) cannot be in the controllability relation and therefore $CONTROL$ returns false.

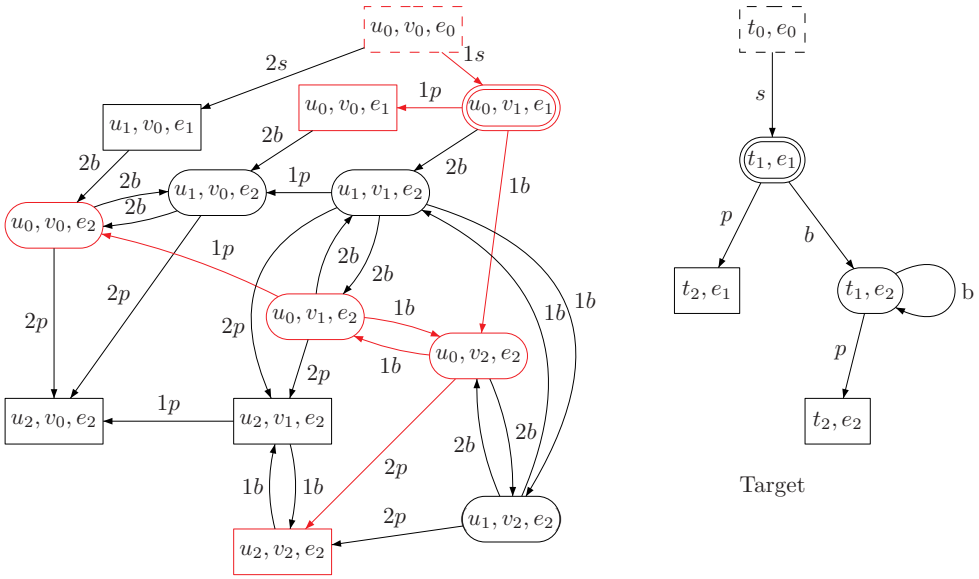
Once the *uncontrollable* transitions are processed, and **all** of them are found to be matched by *May* transitions of the target, the function $CONTROL$ in 5 starts processing the *Must* transitions of the target. But since $Must \subseteq May$ and the target is deterministic then all possible *Must* transitions whose label is in Θ were already processed. Therefore the algorithm processes only the **remaining** $Must$ transitions of the target (line 15 processes $(\Sigma - \Theta)$). This part is done similarly to the case of the Roman Model except that the messages are arbitrarily drawn from a set Com. This difference in messaging is taken care of in the new function $MATCH$ shown in algorithm 6. Given a *Must* of the target $t \xrightarrow{g(e),a} t'$ the $MATCH$ function finds a message α that the orchestrator can send to the community, which causes the community to transition to a new state s', after performing an action a, such that the (t', e', s') is in the controllability relation. But since sending message α *can enable other action*s of the community, the $MATCH$ function checks that those "extra" actions are matched by May transitions of the target (lines 4-7).

3.8 Conclusion

We have proposed a new on-the-fly algorithm to compute the controllability relation for the behavior composition of partially controllable services. The worst-case complexity of the algorithm matches the known lower bound for the problem. However, we believe that in practice it will have better performance because it does not need to explore the full state space to find a solution. Moreover, the algorithm doesn't need to build a priori the composition state space which allows it to handle larger system than solutions using fixpoint techniques. The algorithm is *robust* to

service failure in a sense that it will use the results obtained before failure to speed up the search for a solution after failure.

We also presented an abstraction method that reduced considerably the state space of labelled transition systems. From this abstraction one can infer the non-existence of an solution to the problem from the non-existence of a simulation relation between the abstracted community and the abstracted target. Deciding on the existence of a simulation relation for abstracted systems can be very fast since the abstracted systems are orders of magnitudes smaller than the original systems. Furthermore, when a simulation for the abstracted systems exists it is used to speed up the search of the proposed algorithm. Finally, we presented a version of the on-the-fly algorithm that can find the controllability relation when the target is specified using modal specifications.



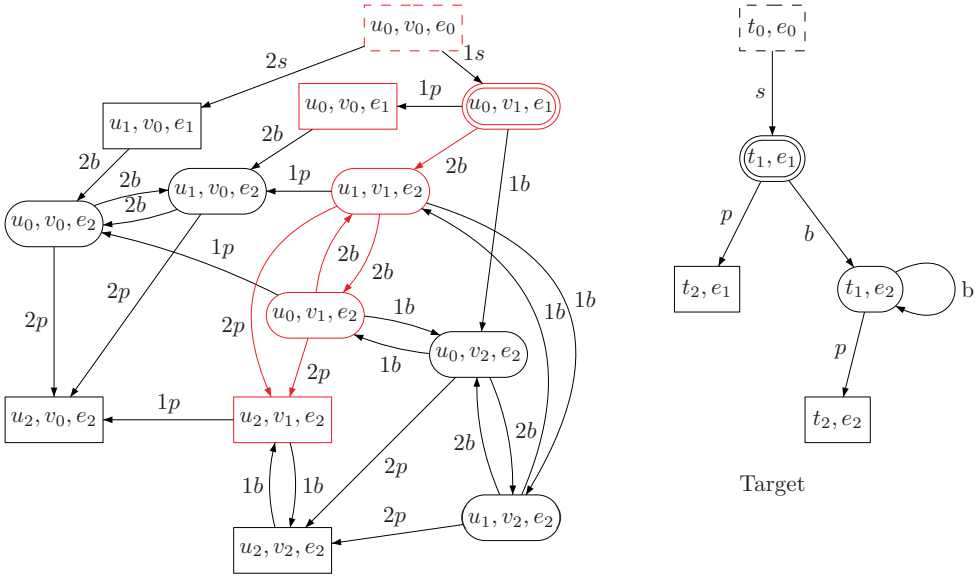
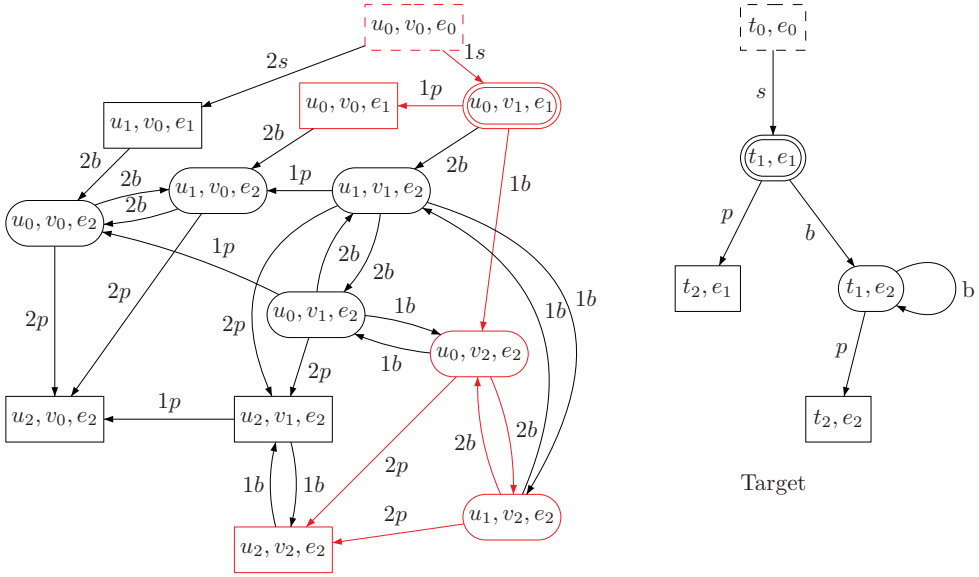
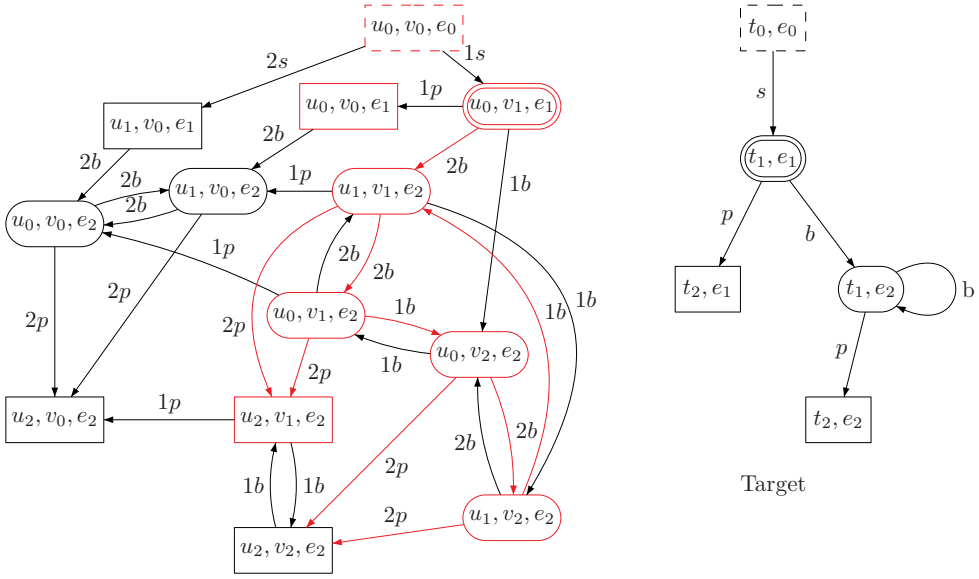


Figure 3.2: Full state space of the community and the target service including solutions 1 and 2. The red nodes and transition denote two possible controllability relations. The transitions search, pay, and buy are shortened to s,p, and b respectively.



Solution 3



Solution 4

Figure 3.3: Full state space of the community and the target service including solutions 3 and 4. The red nodes and transition denote two possible controllability relations. The transitions search, pay, and buy are shortened to s,p, and b respectively.

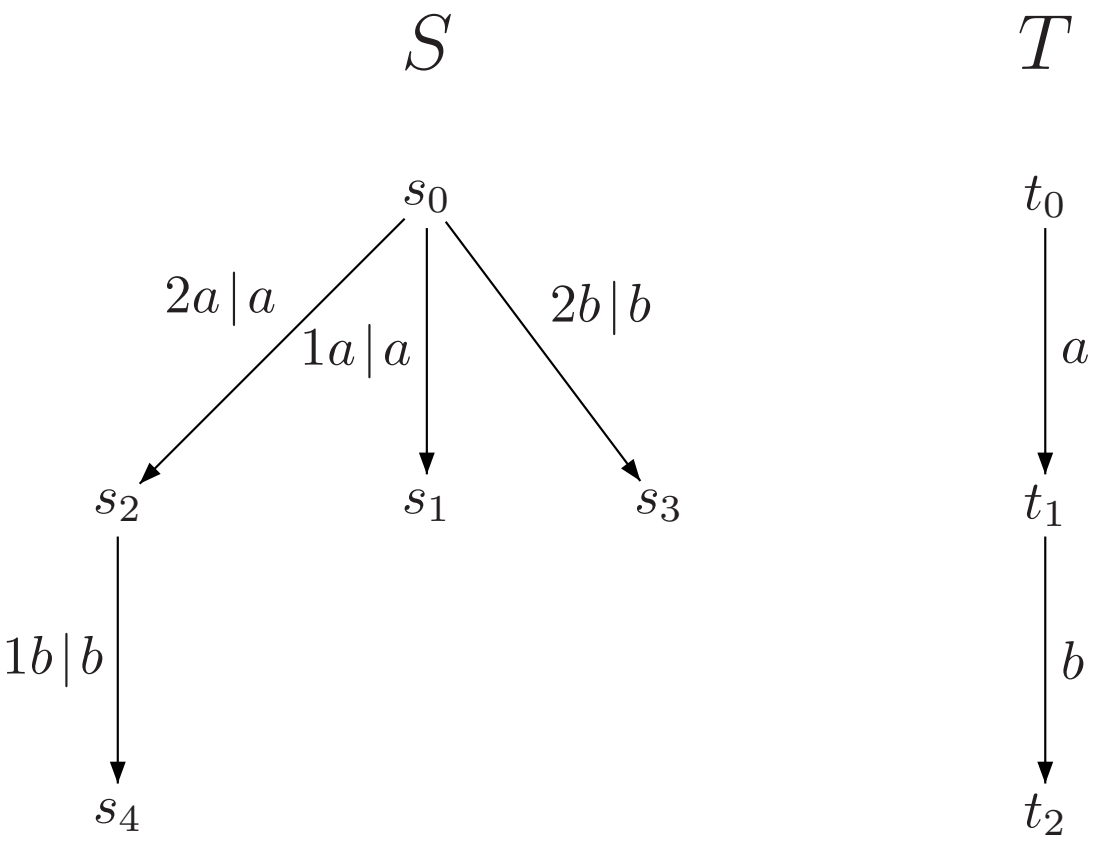


Figure 3.4: A simple example to illustrate the algorithm

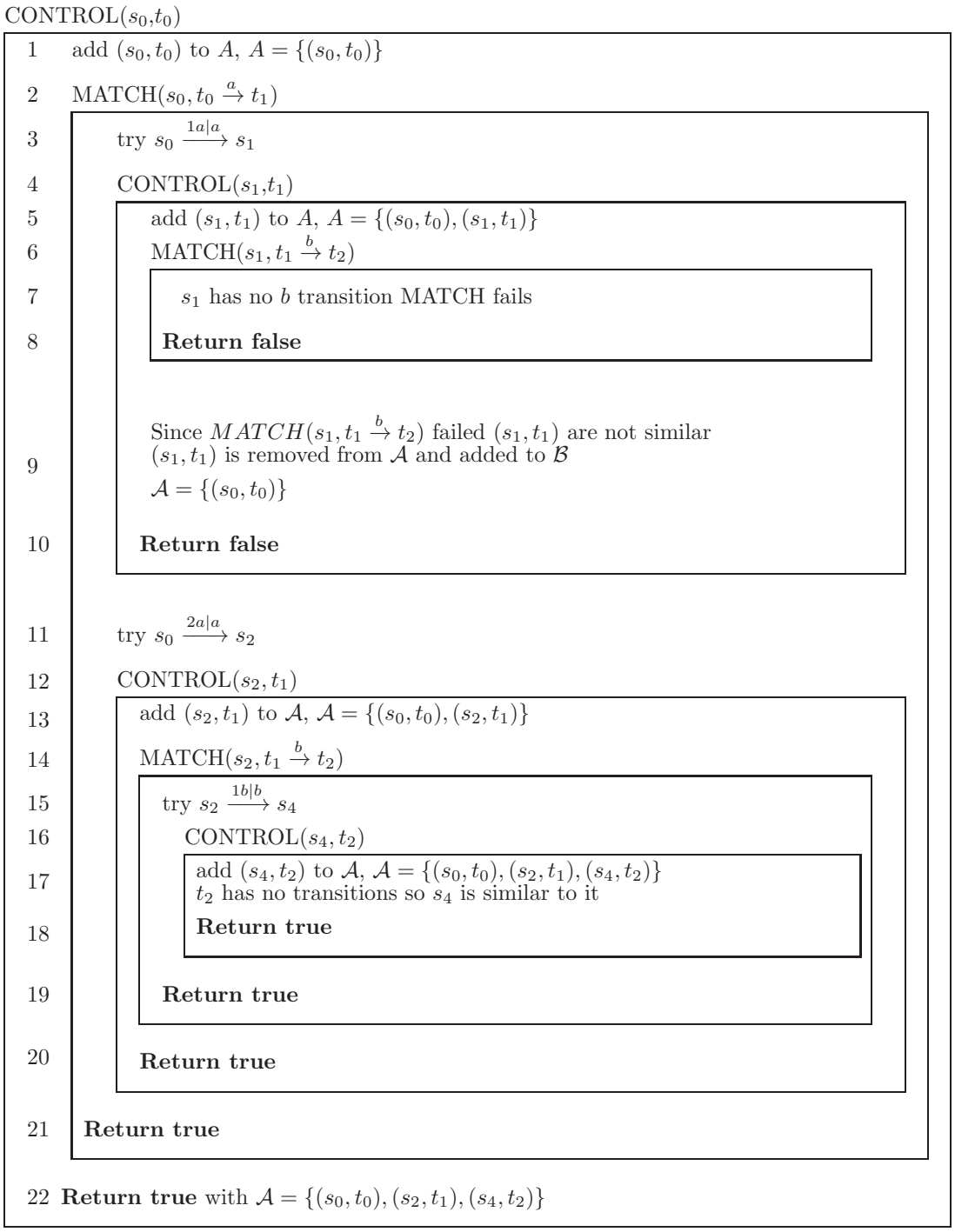


Figure 3.5: The call stack for the simple example shown in Figure 3.4

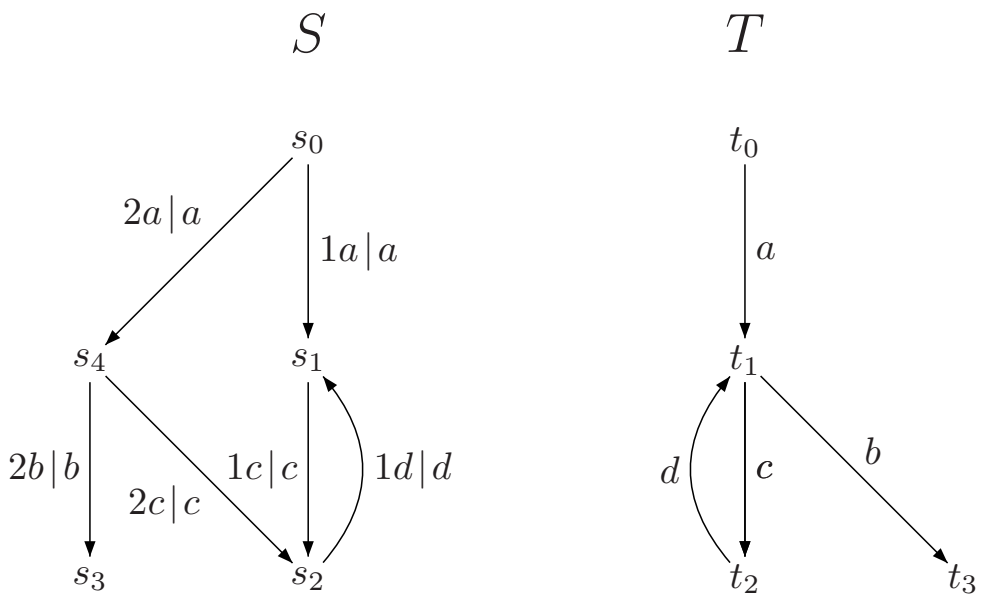


Figure 3.6: Example of a node processing problem

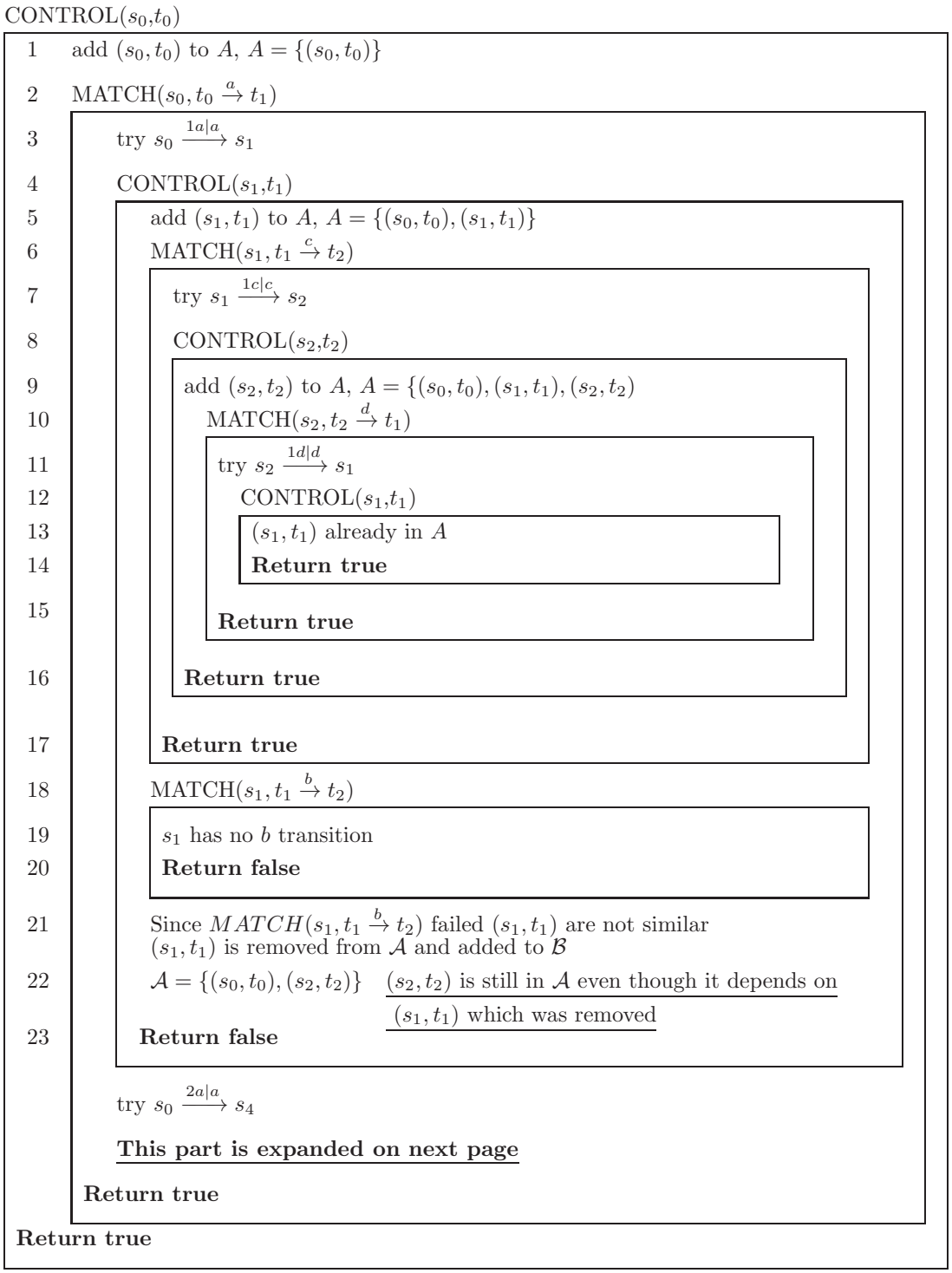


Figure 3.7: Call stack for example in Figure 3.6

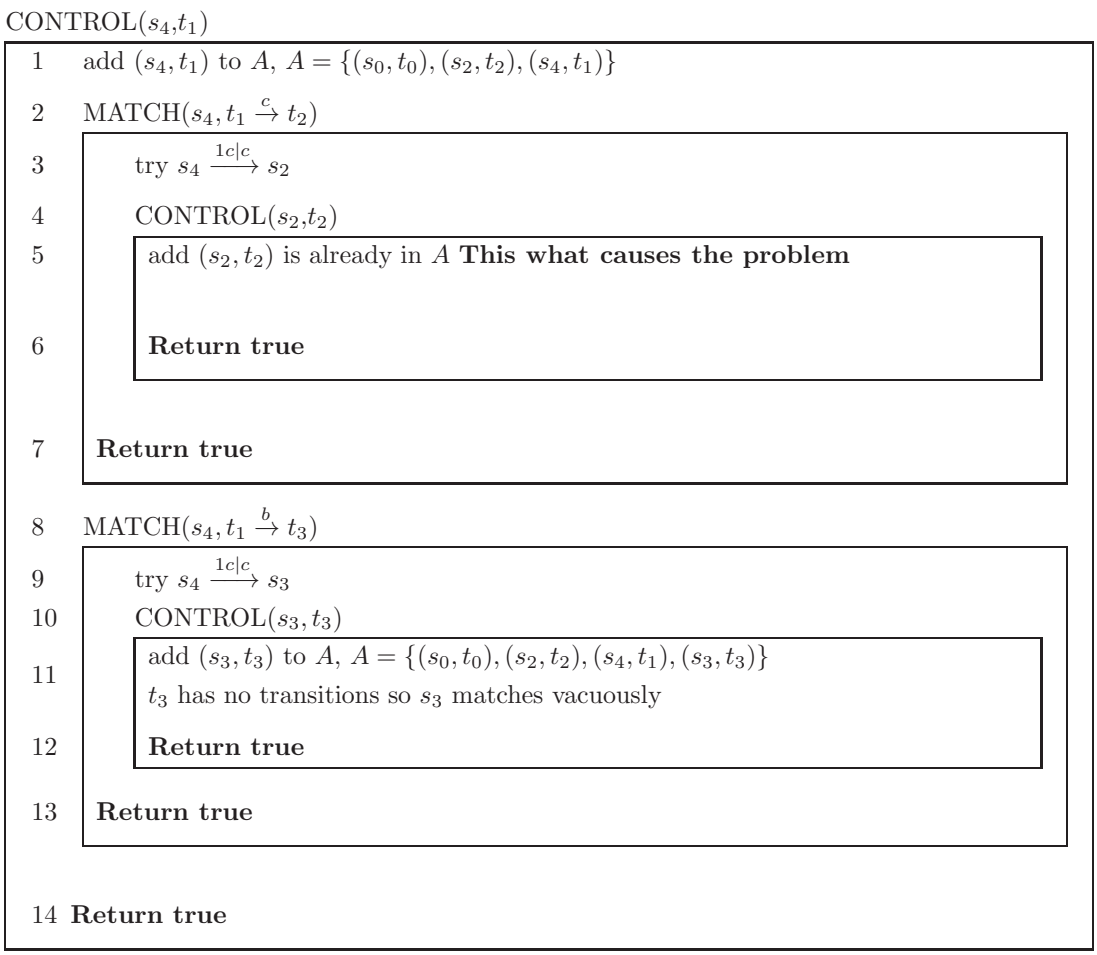


Figure 3.8: Continuation of call stack in Figure 3.7

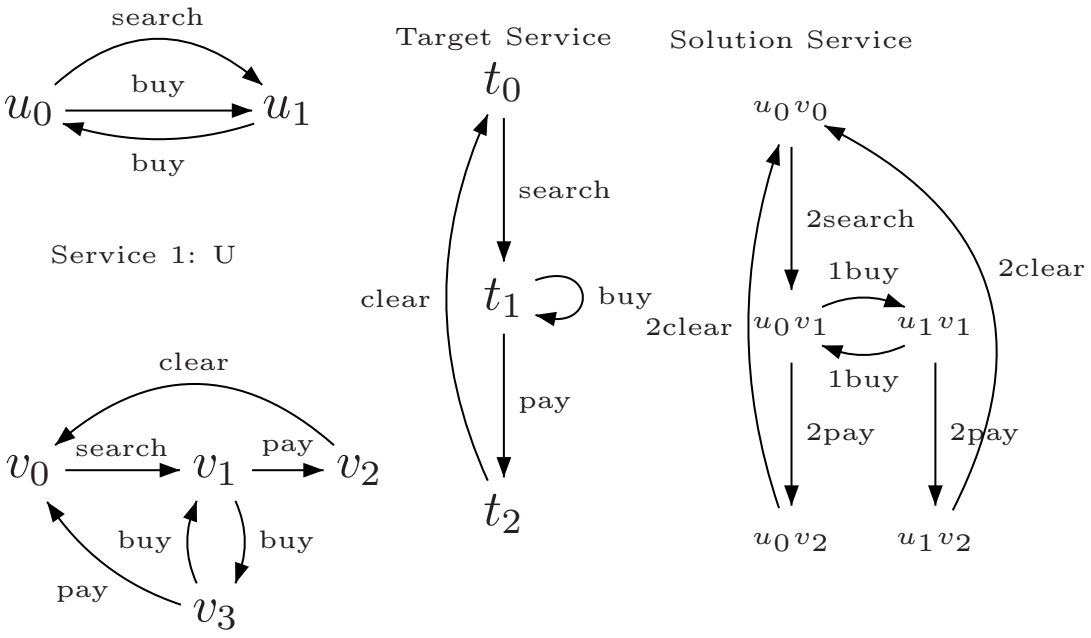


Figure 3.9: Example services together with the solution to be abstracted

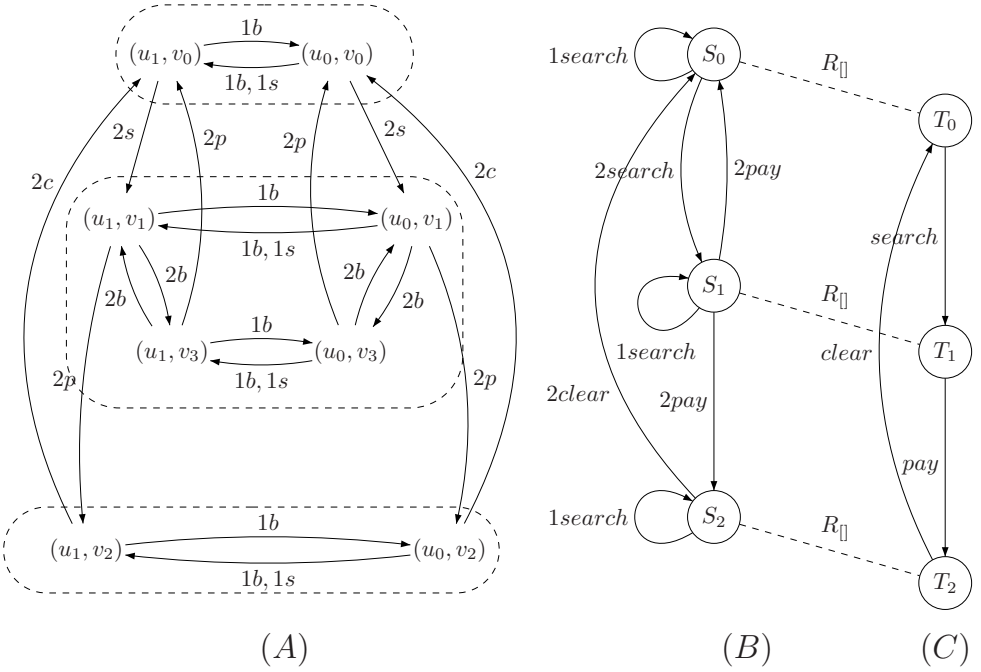


Figure 3.10: (A) is the original community with equivalence classes of the closure relation shown as dashed ovals. (B) is the resulting abstraction. (C) is the abstraction of the target. The relation $R_{[]}$ is a simulation relation

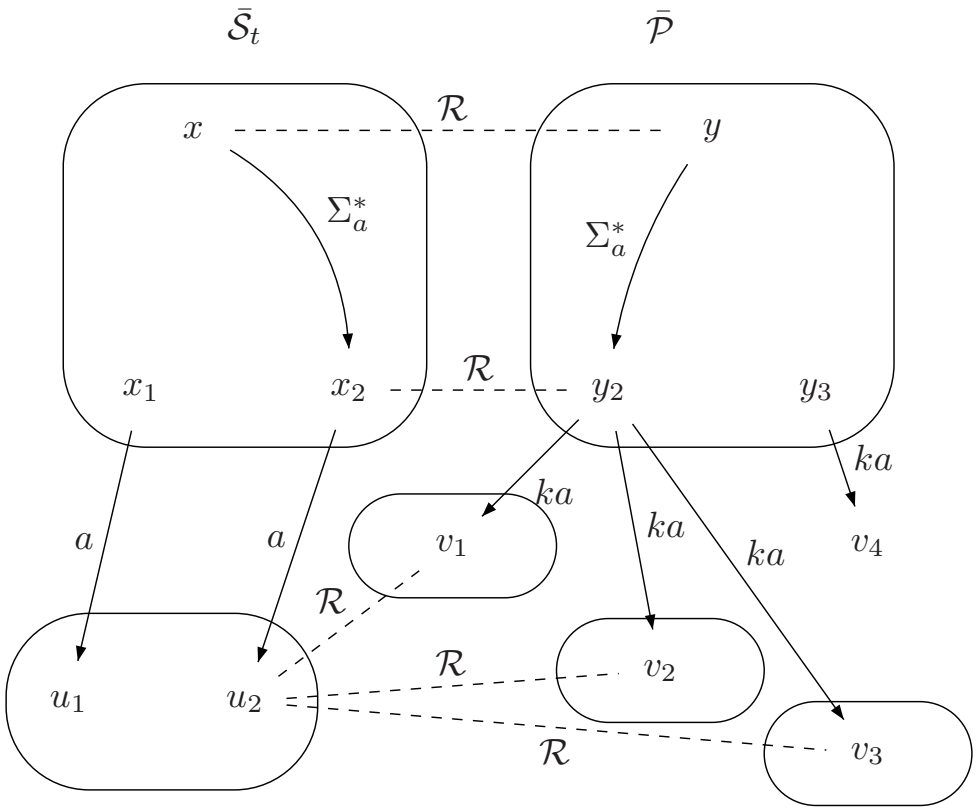


Figure 3.11: Used in proof of theorem 3.6.2. Ovals are the equivalence classes of branching bisimulation for $\mathcal{S}_t$ and closure relation for $\mathcal{S}$. $\mathcal{R}$ is the original controllability relation.

Chapter 4

Orchestration under partial information

Contents

4.1 Introduction

4.1.1 Motivation

In this chapter the case of an orchestrator with partial observation is tackled. In such a situation there are some actions which are uncontrollable similarly to the problem discussed in chapter 3. In addition, the orchestrator does not know exactly in which state the community is in. Recall that we proved in chapter 2 that an orchestrator with partial information exists if and only if a set of observation relation exists. This chapter has two contributions to the behavior composition problem. First we develop a fixpoint algorithm to compute the set of observation relations. We also study the complexity of the algorithm. The second contribution is an on-the-fly algorithm to compute the set of observation relations. The advantage of an on-the-fly algorithm in the case of partial information is even more important than the case of perfect information. In most methods that deal with partial information there is a preliminary first step that computes what is called a *belief state* space for the system under study [GB96] where one keeps track of all states the system could be possibly in after a sequence of actions. *After* the computation of the belief state system, which is typically exponential in the number of services, a solution is obtained using the same method used for systems with perfect information. This means that even if no solution exists such an approach requires the computation of the whole belief state space. By contrast, the on-the-fly algorithm proposed in this chapter, computes the belief states and seeks a solution at the same time. This means that it could find a solution or determine that no solution exists without computing the whole belief state.

As mentioned above, most of the approaches to the composition problem with partial information, such as [GMP09] and [BPT10], are based on the *belief state* concept [GB96]. Our approach for the unobservable actions is similar to [BCF08b] but is different, and more general, than the one discussed in [GMP09] and [BPT10]. First in [GMP09] all actions are taken as observable *and* controllable while each state is assigned a observation value via a given function. In such a model one can infer in which state(s) the community is in by observing what transitions were made. This means that even if two states have the same observation (i.e. indistinguishable) the fact that the sequence of actions taken to reach each of them is observable, makes them distinguishable. Therefore our model extends [GMP09] both in terms of observability and controllability .

The model taken in [BPT10] is similar to ours in that some (internal) actions are not controllable and not observable. The difference is that those actions are considered internal and they don't have to match the target actions whereas in our model these actions should match the target action even if they are uncontrollable/unobservable.

In our model a subset of actions are unobservable and uncontrollable. This is a special case of the controller synthesis problem [RW89], by assuming that the sets of *unobservable* and the sets of *uncontrollable* actions are the same.

4.1.2 Example

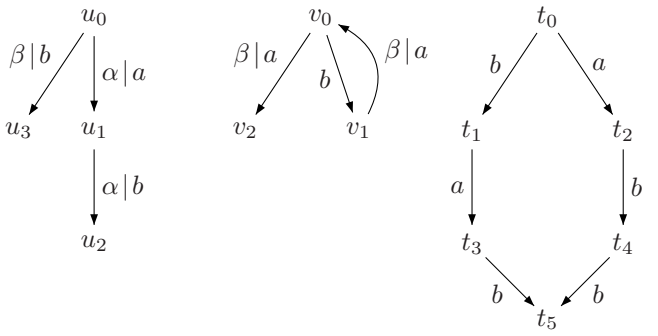


Figure 4.1: Two available services and a target service

The difference in modeling partial information discussed above, is best illustrated with an example. In Figure 4.1 we show two available services U and V and the target service T. The orchestrator can send messages to the community of services (in this case U and V), from a set $Com = \{\alpha, \beta\}$. Note that in this example when the community is in the state (u_0, v_0) and the orchestrator sends a message β either service U makes a $u_0 \xrightarrow{\beta|b} u_3$ transition or service V makes a $v_0 \xrightarrow{\beta|a} v_2$ transition. This is in contrast of the Roman Model [GPS13] where the orchestrator has a finer control, it can direct each service individually on which action to make. In the example of Figure 4.1 the Roman Model admits a solution whereas our model does not. Also in the Roman Model all transitions are controllable. For example the transition $v_0 \xrightarrow{b} v_1$ can also be controlled (i.e. enabled or disabled) whereas in our model this transition cannot be controlled. In summary, the orchestrator in our model "remembers" only the messages it has already sent. In other words, the orchestrator is assumed to observe only its own transitions.

We continue our discussion with a simplified model that *admits* a solution by removing the $u_0 \xrightarrow{\beta|b} u_3$ transition as shown in Figure 4.2. The community state space together with the orchestrator and the solution are shown in Figure 4.3. Initially the orchestrator is in state Ω_ϵ, meaning it has not sent any message yet. We use the special message ϵ to denote that no message was sent. Initially the community can be in any of the states $\{(u_0, v_0)$ or $(u_0, v_1)\}$. Then the observable of those states is the same and is equal to ϵ. Similarly, by sending a message α the controller can cause the state (u_0, v_0) to transition to state (u_1, v_0), and state (u_0, v_1) to transition to state (u_1, v_1). All other states are unaffected. This means that if the orchestrator sends α then the community can be in states $\{(u_1, v_1), (u_1, v_0)\}$. The observable of these two states is α.

On the other hand, in the model used in [GMP09], because it assumes full observation of

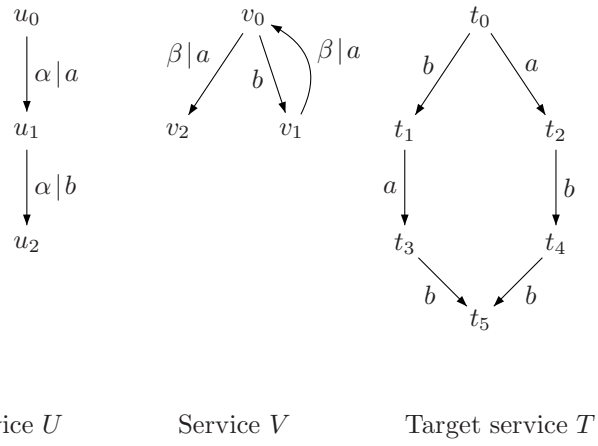


Figure 4.2: A simplified setup that admits a solution

all the actions, the orchestrator can observe the transition $(u_0, v_0) \xrightarrow{b} (u_0, v_1)$ and therefore distinguishes between them. In that model the two states (u_0, v_0) and (u_0, v_1) belong to different belief states whereas in our model because the transition $(u_0, v_0) \xrightarrow{b} (u_0, v_1)$ is unobservable (and uncontrollable) the orchestrator cannot tell if the community is in state (u_0, v_0) or (u_0, v_1).

To compare the two different situations we show one of many possible solutions (i.e. orchestrators) when all the actions are observables in Figure 4.4. In the LTS representing the orchestrator in Figure 4.4, the b transitions are *uncontrollable* but *observable*. This is why the orchestrator changes state, for example $\Omega_0 \xrightarrow{b} \Omega_1$. Note that what is shown is the "most general" solution. For example two different orchestrators (among others) can be obtained from the solution in Figure 4.4, one by removing the transition $\Omega_1 \xrightarrow{\beta}$ another by removing $\Omega_1 \xrightarrow{\alpha}$ because both of them model the same behavior of the target. The important point to make in this example is that because the transition $(u_0, v_0) \xrightarrow{b} (u_0, v_1)$ is *observable* then the orchestrator when in state Ω_1 has the choice of sending either α or β. By contrast, in the case when the b transition is *unobservable*, only the message α will work and that is why there is only a single solution in that case. Finally, it should be noted that the solution in the unobservable case is also a solution when all the transitions are observable. Thus in that sense it is more general.

4.1.3 Definitions

As in Chapter 2 the concept of observation is formalized as follows. Given a community of available services $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ and a target service $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, \delta_t \rangle$ we defined (see definition 2.4.1) the *message trace* function

$$\sigma : S \times \Sigma^* \longrightarrow Com^*$$

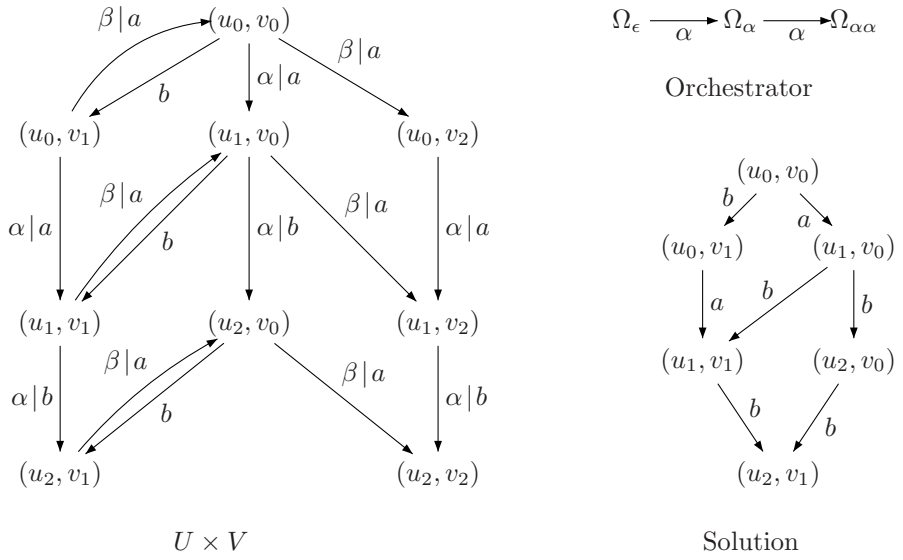


Figure 4.3: Full community space and the solution provided by the shown orchestrator

that returns the observable string given a state in S and a trace in Σ^* where $\Sigma = \Sigma_u \cup \Sigma_t$. For example, in Figure 4.3, $\sigma((u_0, v_0), \epsilon) = \sigma((u_0, v_1), b) = \epsilon$, the empty string, whereas $\sigma((u_1, v_1), ba) = \alpha$ and $\sigma((u_2, v_1), bab) = \alpha\alpha$. Each state of the orchestrator can be represented by (not necessarily unique) string. For example, Ω_0 is represented by the empty string, and Ω_1 is represented by α.

Recall that the transition function for the community under orchestrator with partial information is given by:

$$\delta_\Omega(s, a) = \delta_u(s, a) \bigcup_{\alpha \in Com} \delta_u(s, \alpha \mid a) \odot \Omega(\sigma(s, \tau), \alpha)$$

Where the trace $\tau \in \Sigma^*$ is determined from history of s as computed by the multi-step transition which is defined recursively:

$$\Delta_\Omega(s^0, \epsilon) = \{s^0\}$$

$$\Delta_\Omega(s^0, \tau a) = \bigcup_{s \in \Delta_\Omega(s^0, \tau)} \left[\delta_u(s, a) \bigcup_{\alpha \in Com} \delta_u(s, \alpha \mid a) \odot \Omega(\sigma(s, \tau), \alpha) \right]$$

The composition problem entails the synthesis of an orchestrator such that for any trace τ and for any action a, if $s \in \Delta_\Omega(s^0, \tau)$ and $t \in \Delta_t(t^0, \tau)$ then the following holds

$$\delta_t(t, a) \neq \emptyset \Leftrightarrow \delta_\Omega(s, a) \neq \emptyset$$

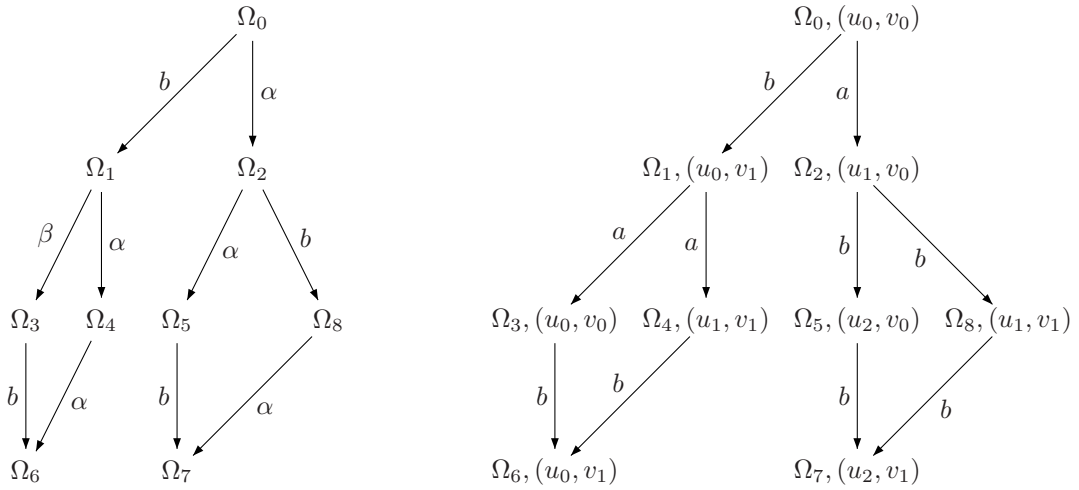


Figure 4.4: One possible orchestrator and solution when all actions are observable

Recall from Chapter 2, theorem 2.4.1, an orchestrator with partial information existence iff there exists a set of observation relations between the community and the target. Furthermore, when such set of relations is obtained the orchestrator's is represented by a labelled transitions system whose states are the set of relations and the transitions are the same as the transition between those relations.

It is convenient to recall explicitly the properties of a set of observation relations from definition 2.4.5. The set of relations $Z \subseteq 2^{S_t \times S}$, is called a set of observation relations iff every $R \in Z$ and every $(t, s) \in R$ has the following two properties:

P1 If $s \xrightarrow{a} s'$ then $\exists t'.t \xrightarrow{a} t' \wedge (t', s') \in R$

P2 If $t \xrightarrow{a} t'$ then

1. Either $\exists s'.s \xrightarrow{a} s' \wedge (t', s') \in R$
2. Or $\exists s', \alpha \in Com, R' \in Z$ with

$$s \xrightarrow{\alpha|a} s' \wedge (t', s') \in R'$$
$$\wedge$$
$$\forall (u, v) \in R(v \xrightarrow{\alpha|b} v' \Rightarrow \exists u'.u \xrightarrow{b} u' \wedge (u', v') \in R')$$

To give a simple and intuitive example of such a set we use Figures 4.2 and 4.3 as a guide.

The set of observation relations for the problem are:

$$R_0 = \{((u_0, v_0), t_0), ((u_0, v_1), t_1)\}$$
$$R_1 = \{((u_1, v_1), t_3), ((u_1, v_0), t_2)\}$$
$$R_2 = \{((u_2, v_0), t_4), ((u_2, v_1), t_5)\}$$

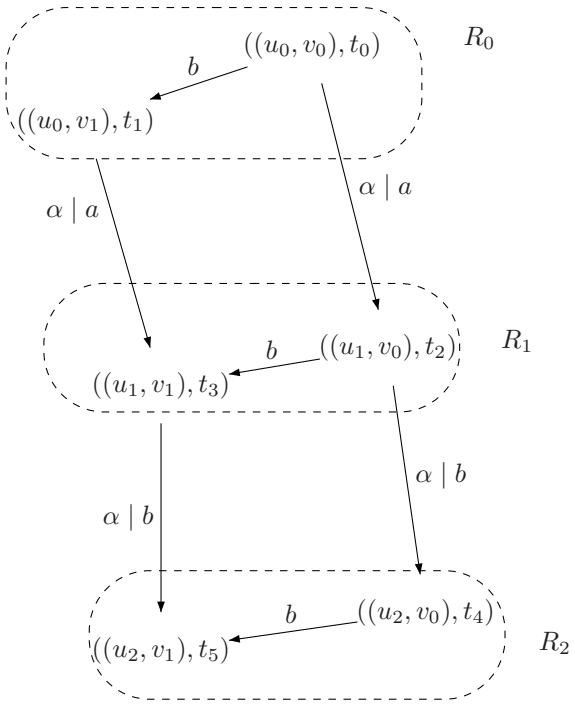


Figure 4.5: Observation relations graph relating the community state space in Figure 4.3 and the target in Figure 4.1

The properties **P1** and **P2** of the observation relations can be explained with the help of Figure 4.5. For property **P1**, one can see that for each relation, if the community makes an uncontrollable (and therefore unobservable) transition then the target can make the same transition such that the community state and target state are also *in the same* relation. For example, in Figure 4.5, consider $((u_0, v_0), t_0) \in R_0$. The transition $(u_0, v_0) \xrightarrow{b} (u_0, v_1)$ is matched by $t_0 \xrightarrow{b} t_1$ such that $((u_0, v_1), t_1) \in R_0$, the same relation.

Now for property **P2**. When the target makes a transition then it is either matched by an uncontrollable community transition, to the same relation as we saw in the example above with the b transitions, or by a *controllable* transition. In this example the target makes an a-transition that is not matched by an uncontrollable community transition. Therefore, the orchestrator needs to find a controllable transition which it enables by sending the message α. Also it should be such

that all the actions enabled by α transition to the same relation. In the above example, consider $((u_0, v_0), t_0) \in R_0$ again. The $t_0 \xrightarrow{a} t_2$ transition has to be matched by a controllable transition. From Figure 4.3 there are two possibilities and only one of them works $(u_0, v_0) \xrightarrow{\alpha|a} (u_1, v_0)$. But by enabling actions prefixed by α the orchestrator will enable also the $(u_0, v_1) \xrightarrow{\alpha|a} (u_1, v_1)$ transition. But this transition is matched by $t_1 \xrightarrow{a} t_3$ and both $((u_1, v_1), t_3)$ and $((u_1, v_0), t_2)$ are in the *same* relation R_1. Clearly the set $\{R_0, R_1, R_2\}$ has the properties of a set of observation relations. Note that the relations represent the states of the orchestrator and it is easy to extract the transitions of the orchestrator from the observation relation graph shown in Figure 4.5.

Having recalled the definition of the set of observation relations and how the orchestrator is extracted from them, the remainder of this chapter is concerned with finding a set of observation relations for a given problem.

First we present in section 4.2 a fixpoint algorithm to compute the set of observation relation. Then an on-the-fly algorithm to compute the relations is given in section 4.3.

Algorithm 7: CLOSED returns true iff the input relation is closed with respect to uncontrollable actions

```
1 CLOSED(R)
2 foreach (s,t) ∈ R do
3 |   foreach s -a-> s' do
4 |   |   if t -a-> t' then
5 |   |   |   if (s',t') ∉ R then
6 |   |   |   |   return false
7 |   |   else
8 |   |   |   return false
9 return true
```

4.2 Fixpoint algorithm

In this section we present a fixpoint algorithm to compute the largest set of observation relations for a given target service and community of available services. The argument that a largest such set exists follows directly from casting the problem as a fixpoint of some function as done in this section. This is similar to the algorithm that was presented in the previous chapter for the case of perfect information.

4.2.1 Algorithm

The fixpoint algorithm presented in this section depends on the concept of closed relations which we introduce next.

Definition 4.2.1. *Let $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ be a community of available services and $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, \delta_t \rangle$ be a target service. A relation $R \subseteq S \times S_t$ is said to be closed with respect to uncontrollable actions iff:*

$$\forall (s,t) \in R, \forall a \in \Sigma_u : s \xrightarrow{a} s' \Rightarrow \exists t'.t \xrightarrow{a} t' \wedge (s',t') \in R$$

The above definition basically takes care of the uncontrolled transitions. It should be noted that any relation in the set of observation relations we are seeking should be closed. A simple algorithm based on definition 4.2.1 to test whether a given relation is closed is given in algorithm 7. What remains is to synthesize an orchestrator to control the controllable actions. Toward this end we use an iterative fixed point algorithm similar to the case of perfect information. This algorithm belongs also to the category we called "fixpoint" because the starting point is all the set of all subsets of $S \times S_t$.

$$Z_0 = 2^{S \times S_t}$$

Next the set of all closed relations in Z_0 is computed:

$$Z_1 = \{R \in Z_0 \mid CLOSED(R)\}$$

Algorithm 8: Computing Z_1

```
1 Z_1 ← ∅
2 foreach R ⊆ S × S_t do
3 |   if CLOSED (R) then
4 |   |   Z_1 ← Z_1 ∪ {R}
5 return Z_1
```

The computation of the set Z_1 is shown in algorithm 8. The braces in line 4 in the algorithm are used to stress the fact that a closed relation R is an *element* of Z_1. Clearly if $Z_1 = \emptyset$ then no set of observable relations exists and therefore the problem has no solution. Only when $Z_1 \neq \emptyset$ the method proceeds to find the set of observation relations. Any relation in $R \in Z_1$ satisfies, by construction, property **P1** of the set of observation relation, namely that it is closed with respect to uncontrollable actions. Define a function over the set of sets of relations as:

$$
\begin{aligned}
F(Z_i) = \{R \in Z_i \mid \forall (s,t) \in R : \\
& t \xrightarrow{a} t' \Rightarrow \Big(\exists s'.s \xrightarrow{a} s' \wedge (s',t') \in R\Big) && \text{(4.1a)}\\
& \vee \\
& \Big(\exists \alpha \in Com, s' \in S, R' \in Z_i.s \xrightarrow{\alpha|a} s' \wedge (s',t') \in R' && \text{(4.1b)}\\
& \qquad \wedge \\
& \qquad \forall (u,v) \in R : u \xrightarrow{\alpha|b} u' \Rightarrow v \xrightarrow{b} v' \wedge (u',v') \in R' \quad \Big) && \text{(4.1c)}
\end{aligned}
$$

To compute the set of observation relation the algorithm applies repeatedly function F starting with the set of relations Z_1. Therefore the algorithm computes the sequence:

$$Z_{i+1} = F(Z_i)$$

It is clear from the definition of the function F that for any set of relations Z_i, we have $F(Z_i) \subseteq Z_i$. Therefore for a given iteration, relations are *removed* but never *added*. Since the algorithm's starting point is Z_1, and by construction, all relations in Z_1 satisfy property **P1** then for any iteration, any relation in $F(Z_i)$ satisfies property **P1**. This is why in the definition of the function F there is no explicit check for the matching of uncontrollable community transitions by target transition.

On the other hand Z_0 and therefore Z_1 is finite then the above procedure terminates after a finite number of steps, say j. Then $F(Z_j) = Z_j$ and the set of relations Z_j has, by construction of Z_1 and F, the properties of a set of observation relations.

In this section we presented a fixpoint procedure to compute the set of observation relation. In the next section we analyse the complexity of this algorithm.

4.2.2 Complexity

We compute an upper bound for the algorithm discussed in the previous section. First we compute the cost of a single iteration $Z_{i+1} = F(Z_i)$. From equation (4.1) one can see that each relation $R \in Z_i$ is processed to decide whether it will be kept in Z_{i+1}. Let s (t) be a community (target) state. The number of a transitions that s (t) can make is denoted by $|\, s \xrightarrow{a}|$ ($|\, t \xrightarrow{a}|$). Then the cost of processing a single relation R in Z_i can be written as:

$$\sum_{(s,t)\in R} \sum_{a\in\Sigma} \sum_{\alpha\in Com} |\, t \xrightarrow{a}| \cdot \left(|\, s \xrightarrow{a}| + |\, s \xrightarrow{\alpha|a}| \cdot \sum_{(u,v)\in R} \sum_{b\in\Sigma} |\, u \xrightarrow{\alpha|b}| \right)$$

The above cost can be explained as follows. For every $t \xrightarrow{a}$ transition, we need to find a matching a transition for s either uncontrollable, $s \xrightarrow{a}$, or controllable $s \xrightarrow{\alpha|a}$. In the case of controllable action with message α, one also needs to check *all* transitions in R enabled by that message, hence the last sum in the above cost. Clearly the above cost is denominated by the double sum. Keeping only the double sum and taking into consideration that the target is deterministic (i.e for a given a, $|\, t \xrightarrow{a}| = 1$) then the cost of processing a single relation R in the computation of $F(Z_i)$ becomes:

$$\begin{aligned}
&\sum_{(s,t)\in R}\sum_{a\in\Sigma}\sum_{\alpha\in Com} |\, s \xrightarrow{\alpha|a}| \cdot \sum_{(u,v)\in R}\sum_{b\in\Sigma} |\, u \xrightarrow{\alpha|b}| \\
&\leq \left(\sum_{(s,t)\in R}\sum_{a\in\Sigma}\sum_{\alpha\in Com} |\, s \xrightarrow{\alpha|a}|\right) \cdot \left(\sum_{(u,v)\in R}\sum_{\beta\in Com}\sum_{b\in\Sigma} |\, u \xrightarrow{\beta|b}|\right) \\
&= |\, L_R\,|^2
\end{aligned}$$

Where $|\ L_R\ |$ is the number of controllable transitions in R. Having computed the cost of processing a single relation, the cost of iteration i of the algorithm is the sum over all relations present at iteration i:

$$F(Z_i) = O\left(\sum_{R\in Z_i} |\, L_R\,|^2\right)$$

The total cost of the algorithm is then

$$O\left(\sum_{i=1}^{j}\sum_{R\in Z_i} |\, L_R\,|^2\right)$$

Where j is the iteration in which we reach a fix point :$Z_j = F(Z_j)$. First we give an upper bound for $|\, L_R\,|$. Let R be an arbitrary relation and $(s,t) \in R$ an arbitrary element. For a given $\alpha \in Com$ and $a \in \Sigma$, a pair (s,t) can connect at most to all other pairs in $S \times S_t$. Therefore $|\, s \xrightarrow{\alpha|a}| = O(|\, S \times S_t\,|)$. This implies that $|\, L_R| = O(|\ R\ | \cdot |\, S \times S_t\,|)$ where $|\ R\ |$ is the number of elements in R. Then an upper bound for complexity of the algorithm would be

$$O\left(|\, S \times S_t\,|^2 \sum_{i=1}^{j}\sum_{R\in Z_i} |\, R|^2\right)$$

Let $N = |\, Z_1\,|$. Since F is monotone decreasing then $Z_1 \supset \ldots, \supset Z_j$. The worst case for the algorithm occurs when $Z_j = \emptyset$, i.e. the problem has no solution. Furthermore, in the worst case Z_i and Z_{i+1} differ by only one set, i.e. $|\, Z_i| = |\, Z_{i+1} + 1\,|$. Adding these two properties we get that

in the worst case it takes $|\,Z_1\,|$ iterations for the algorithm to determine that there is no solution. In addition $Z_1 \supset Z_i$ for all i then the complexity becomes:

$$\begin{aligned} & O\left(|\,S \times S_t\,|^2 \sum_{i=1}^{|Z_1|} \sum_{R \in Z_i} |\,R|^2\right) \\ &= O\left(|\,S \times S_t\,|^2 \sum_{i=1}^{|Z_1|} \sum_{R \in Z_1} |\,R|^2\right) \\ &= O\left(|\,S \times S_t\,|^2 |\,Z_1\,| \sum_{R \in Z_1} |\,R|^2\right) \end{aligned}$$

The expression $\sum_{R \in Z_1} |\,R\,|^2$ is the sum of the square of number of elements in all relations in Z_1. The above complexity depends on the size of Z_1 which is at most equal to the size of $Z_0 = 2^{S \times S_t}$. Let R_k be a subset of $S \times S_t$ (i.e. an element of Z_0) with size k. Clearly the range of k is from zero for the empty set up to $N =|\,S \times S_t\,|$ for $S \times S_t$ itself. The number of subsets of $S \times S_t$ of size k is given by the binomial coefficient $\binom{N}{k}$. Therefore the sum can be written as

$$\begin{aligned} \sum_{R \in Z_1} |\,R|^2 &\leq \sum_{R \in Z_0} |\,R|^2 \\ &= \sum_{k=0}^{N} k^2 \binom{N}{k} \\ &= (N + N^2) 2^{N-2} \end{aligned}$$

Finally the complexity of the fixpoint algorithm has the following upper bound

$$O\left(N^4 \cdot 2^{2N}\right)$$

Of interest is the dependence of the complexity on the number of services. For n services the numbers of states is $S = S_1 \times S_2 \ldots S_n$ where S_i is the set of states of service i. Recall that $N =|\,S \times S_t\,|$ thus $N = N_1 \cdot N_2 \ldots N_n \cdot N_t$ where N_t is the number of states of the target service. Since N is exponential in the number of services then the complexity of the above algorithm is 2EXPTIME in the number of services. Obviously this is not a tight bound since many assumptions were made. Also the number of iterations can be optimized as in the case of the Paige-Tarjan algorithm [PT87] but the 2EXPTIME behavior would still be there.

In view of the large complexity of the global algorithm it is important to develop an algorithm that is amenable to heuristics that could reduce its complexity. In particular, the case when no orchestrator exists should be done faster than by the above algorithm. Toward that end an on-the-fly algorithm, similar to the case of orchestrator with perfect information, is developed in the next section.

Algorithm 9: On the fly algorithm

```
1  clos ← CLOSURE((∅, s^0, t^0))
2  if clos = ∅ then
3  |  return ∅
4
5  changed ← true
6  while changed = true do
7  |  changed ← false
8  |  Z ← ∅
9  |  res = CONTROL(clos)
10 |  if res = false then
11 |  |  break
12 return Z
```

4.3 On-the-Fly Algorithm

In this section we present an algorithm to construct incrementally a set of observation relation by constructing the relations when needed. This incremental approach is more efficient than computing all the subsets from the start.

4.3.1 Algorithm

The goal of the on-the-fly algorithm presented in this section is to compute a set of observation relations, Z. Since a necessary condition for a relation R to belong to Z is that R is closed, we define a procedure that computes the closure of a single pair. This allows us to determine if a given pairs belongs to closed set or not. The closure operation is performed in the $CLOSURE$ procedure which is defined in an inductive manner and shown in Algorithm 10. It returns $\emptyset$ if the pair (s,t) does not belong to a closed set, and the closure of (s,t) otherwise. Note that the union of closed sets is also a closed set. The function $CLOSURE$ basically checks if a pair (s,t) belongs to a closed set as defined in definition 4.2.1.

The function $CONTROL(R)$ returns true if the relation R belongs to the sought set of relations Z, otherwise it returns false. As we have already seen, a necessary condition for a relation R to be in Z is that R is closed. This way the relation passed as a parameter to $CONTROL$ is always closed. For this to be true the algorithm has to start with closure of the initial state (s^0, t^0) where s^0 is the initial state of the community and t^0 is the initial state of the target. After the initial call, the input to the function $CONTROL$ is always a closed set because $CONTROL$ is called from $MATCH$ (line 10) and passes it the set R' as a parameter where R', if passed to $CONTROL$, is guaranteed to be closed by the function $CHECK$ (see lines 6-10 in function $MATCH$). The overall functioning of the algorithm is shown in Algorithm 9.

As in the case of perfect information the algorithm adds relations to Z in preorder and checks

Algorithm 10: Function CLOSURE

```
1  CLOSURE(R,(s,t))
2  if (s,t) ∈ bad then
3  |  return ∅
4  if (s,t) ∈ R then
5  |  return R
6  R' ← R ∪ {(s,t)}
7  foreach s -a-> s' do
8  |  if t -a-> t' then
9  |  |  R' ← CLOSURE(R', (s',t'))
10 |  |  if R' = ∅ then
11 |  |  |  bad ← bad ∪ {(s',t')}
12 |  |  |  return ∅
13 |  else
14 |  |  bad ← bad ∪ {(s,t)}
15 |  |  return ∅
16 return R'
```

them in postorder fashion. As we have seen in chapter 3 this can cause a problem when a relation R depends on another relation R' and R' is removed from Z *after* R was checked. Similar to the case of perfect information we include a variable *changed* which is set to true whenever a relation is removed from Z. The removal of a relation R' from Z can affect the status of some other relation $R \in Z$ that depended on R' and potentially invalidating the inclusion of R in Z. This is why the algorithm is rerun every time, with Z empty, the value of *changed* is true. Once *changed* is false we know that the algorithm has converged. Clearly when $CONTROL$ returns false no set of relations Z exists regardless of the value of *changed*. The aforementioned algorithm assumes that (s^0, t^0) is closed, or alternatively $clos \neq \emptyset$ in the first line of the algorithm, otherwise Z is empty and there is no set of observation relations between the community and the target and thus the problem has no solution. Next we describe the working of each component of the algorithm.

The function $CONTROL$ takes a relation R as a parameter and returns true if for each (s,t) in R and for each target transition it finds a matching community transition. A matching transition for a target transition $t \xrightarrow{a} t'$ is defined by equation (4.1). The actual matching is done by function $MATCH$ described later. In order not to enter in an infinite loop it keeps two sets Y and Z. The set Z is the set of observation relations we are seeking. The set Y contains all the relations that cannot be in Z. It should be noted that Z is set to $\emptyset$ for every iteration of the algorithm whereas Y is not since a set $R \not\subset Z$ for some iteration cannot be $R \subset Z$ in a subsequent iteration.

As mentioned above the match of each target transition is done via the function $MATCH$.

Algorithm 11: Function CONTROL

```
1  CONTROL(R)
2  if R ∈ Y then
3  |  return false
4  if R ∈ Z then
5  |  return true
6  Z ← Z ∪ {R}
7  foreach (s,t) ∈ R do
8  |  foreach t -a-> t' do
9  |  |  res = MATCH (R,s,t -a-> t')
10 |  |  if res = false then
11 |  |  |  GOTO Exit
12 return true
13 Exit: Z ← Z − R
14 Y ← Y ∪ R
15 changed ← true
16 return false
```

By definition the set of relations Z satisfies $F(Z) = Z$ where the function F is the iterative procedure defined in equation (4.1). We rewrite the iterative function symbolically as

$$F(Z_i) = \{R \in Z_i \mid \forall (s,t) \in R : MATCH(R, s, t \xrightarrow{a} t')\} \tag{4.2}$$

Therefore the $MATCH$ procedure checks, with the help of the $CHECK$ auxiliary function, for the first and second part of the disjunction in the iteration in equation (4.1). The first part, i.e. (4.1a) of the disjunction is tested in lines 2-3 in $MATCH$. The second part in the disjunction, i.e. parts (4.1b) and (4.1c), is tested in lines 4-12 in $MATCH$. The first part is straightforward whereas the second part needs some explanation.

If the transition of the target $t \xrightarrow{a} t'$ is not matched by an *uncontrollable* community transition then the orchestrator needs to find an appropriate message $\alpha \in Com$ to send to the community such that:

1. The community can make an $s \xrightarrow{\alpha|a} s'$ transition such that $(s', t') \in R'$ for some **closed** set R'.

2. Any potential transition caused by the message α should be to the same set R' with the condition that R' is also a closed set hence the need for the auxiliary function $CHECK$.

3. Finally the resulting set R', if it is non-empty then it is guaranteed by the $CHECK$ function to be closed, is passed to $CONTROL$ to test whether it belongs to Z.

Algorithm 12: Function MATCH

```
1  MATCH (R, s, t --a--> t')
2  if s --a--> s' then
3  |  return true
4  foreach α ∈ Com do
5  |  if s --α|a--> s' then
   |  |  /* check to see if this α leads to a closed set                */
6  |  |  R' ←CHECK (R, α)
7  |  |  if R' = ∅ then
   |  |  |  /* try different α this one leads to non-closed set         */
8  |  |  |  Continue
9  |  |  else
10 |  |  |  res =CONTROL (R')
11 |  |  |  if res = true then
12 |  |  |  |  return true
   |  /* tried all α's.  none worked                                     */
13 return false
```

The first condition is the existence of $\alpha \in Com$ and $s' \in S$ such that $s \xrightarrow{\alpha|a} s'$. This part is checked in line 5 in $MATCH$. The remaining conditions are checked by the $CHECK$ function. Namely, $CHECK$ checks to see that by enabling transitions with α precondition, that for every (u, v) in R if the community makes some transition $u \xrightarrow{\alpha|b} u'$ then the target can make it also. In other words we don't want to enable actions that can be performed by the community but not by the target. If that is the case then the chosen message α is not correct.

The $CHECK(R, \alpha)$ function returns an empty set, $\emptyset$, if one of the conditions is true

1. Either some $(u, v) \in R$ is such that $u \xrightarrow{\alpha|b} u'$ and $\nexists v'.v \xrightarrow{b} v'$.

2. Or some $(u, v) \in R$ is such that $u \xrightarrow{\alpha|b} u' \wedge v \xrightarrow{b} v'$ but $CLOSURE(u, v) = \emptyset$.

Since the union of closed sets is also a closed set then the set returned by $CHECK$ is either the empty set, $\emptyset$ or if none of the above two situations exits then the return value of $CHECK$ is

$$\bigcup_{\substack{(u,v)\in R \\ u \xrightarrow{\alpha|b} v'}} CLOSURE(u, v)$$

Algorithm 13: Function CHECK

```
1  CHECK (R, α)
2  R' ← ∅
3  foreach (u, v) ∈ R do
4      if u -α|b-> u' then
5          if v -b-> v' then
6              clos = CLOSURE(∅, u', v')
7              if clos = ∅ then
8                  return ∅
9              else
10                 R' ← R' ∪ clos
11         else
12             return ∅
13 return R'
```

4.3.2 Correctness

From Algorithm 11 we see that a relation R is added to the set Z iff the function $CONTROL(R)$ returns true. By inspecting the code for function $CONTROL(R)$ one can see that it returns true when R is found to have the properties:

$$\forall (s,t) \in R, \forall a \in \Sigma :$$
$$t \xrightarrow{a} t' \Rightarrow MATCH(R, s, t \xrightarrow{a} t')$$

Where the clause $MATCH(R, s, t \xrightarrow{a} t')$ means the function $MATCH$ returned the value *true*. We proceed by expanding the call to the $MATCH$ function into its components:

$$\forall (s,t) \in R, \forall a \in \Sigma :$$
$$t \xrightarrow{a} t' \Rightarrow \left(\exists s'.s \xrightarrow{a} s'\right) \vee$$
$$\left(\exists \alpha \in Com, s' \in S.s \xrightarrow{\alpha|a} s' \wedge (CHECK(R,\alpha) \neq \emptyset) \wedge CONTROL(CHECK(R,\alpha))\right)$$

Because the relations passed as parameters to $CONTROL$ and $MATCH$ are closed relations then the first part of the above disjunction: $\exists s'.s \xrightarrow{a} s'$ can be rewritten as

$$\exists s'.s \xrightarrow{a} s' \wedge (s', t') \in R$$

Breaking down the above further by using the construction of the $CHECK$ function and the fact that $CONTROL(R)$ returns true iff $R \in Z$ we get:

$$
\begin{aligned}
&\forall (s,t) \in R, \forall a \in \Sigma : \\
&t \xrightarrow{a} t' \Rightarrow \Big(\exists s'.s \xrightarrow{a} s' \wedge (s',t') \in R\Big) \text{ // line 2-3 in MATCH} \\
&\vee \\
&\Big\{\exists \alpha \in Com, s' \in S. \Big[s \xrightarrow{\alpha|a} s' \wedge \text{ // line 5 in MATCH} \\
&\quad \forall (u,v) \in R \Big(u \xrightarrow{\alpha|b} u' \Rightarrow \exists v'.v \xrightarrow{b} v' \wedge closure(u',v')\Big) \text{ //CHECK on line 6 in MATCH} \\
&\quad \wedge \bigcup_{\substack{(u,v)\in R \\ u \xrightarrow{\alpha|b} u' \\ v \xrightarrow{b} v'}} closure(u',v') \in Z \Big]\Big\} \text{ //CONTROL on line 10 in MATCH}
\end{aligned} \tag{4.3}
$$

Define the relation R' by

$$
R' = \bigcup_{\substack{(u,v)\in R \\ u \xrightarrow{\alpha|b} u' \\ v \xrightarrow{b} v'}} closure(u',v')
$$

Since each $closure(u',v')$ is a closed relation and the union is also closed then R' is a closed relation which also from the construction of equation (4.3) is also in Z. Using the shorthand R' in (4.3) we get that all relations $R \in Z$ satisfy the following property:

$$
\begin{aligned}
&\forall (s,t) \in R, \forall a \in \Sigma : \\
&t \xrightarrow{a} t' \Rightarrow \Big(\exists s'.s \xrightarrow{a} s' \wedge (s',t') \in R\Big) \\
&\vee \\
&\Big\{\exists \alpha \in Com, s' \in S, R' \in Z. \Big[s \xrightarrow{\alpha|a} s' \wedge (s',t') \in R' \\
&\quad \forall (u,v) \in R \Big(u \xrightarrow{\alpha|b} u' \Rightarrow \exists v'.v \xrightarrow{b} v' \wedge (u',v') \in R'\Big)
\end{aligned} \tag{4.4}
$$

By comparing (4.4) and of (4.1) one can see that Z is a fixed point (not necessarily the largest) of (4.1) and therefore the algorithm is correct. Clearly all the above depends on the fact that once a given relation R is found to be in Z no R' which R depends on as shown in equation (4.3) is removed from Z. This is why it is essential that every time a relation R' is removed from Z the algorithm needs to be rerun again.

4.3.3 Complexity

To compute the complexity of the algorithm we start with a single call to $CONTROL$. The complexity for a single call to $CONTROL(R)$ for a given relation R is

$$\sum_{(s,t)\in R}\sum_{a\in\Sigma} \mid t \xrightarrow{a}\mid \cdot \mid MATCH(R,s,t\xrightarrow{a}t') \mid$$

Because the target service is deterministic then $\mid t \xrightarrow{a}\mid \leq 1$. Also the cost of the $MATCH$ function is dominated by the controllable transitions thus

$$\sum_{(s,t)\in R}\sum_{a\in\Sigma}\sum_{\alpha\in Com} \mid s \xrightarrow{\alpha|a}\mid \cdot \mid CHECK(R,\alpha) \mid$$

taking into account the cost of the $CHECK$ function we get

$$\sum_{(s,t)\in R}\sum_{a\in\Sigma}\sum_{\alpha\in Com} \mid s \xrightarrow{\alpha|a}\mid \cdot \sum_{(u,v)\in R}\sum_{b\in\Sigma} \mid u \xrightarrow{\alpha|b} u' \mid \cdot \mid v \xrightarrow{b} v' \mid \cdot \mid closure(\emptyset,u',v') \mid$$

In general the value of $closure(\emptyset,u',v')$ is difficult to compute since it depends on the unknown (u',v') but we can provide an upper bound. First note that the closure function visits a pair once so at most it will visit all the pairs in $S\times S_t$ thus

$$\begin{aligned} \mid closure(\emptyset,u',v') \mid &\leq \sum_{(u,v)\in S\times S_t}\sum_{a\in\Sigma} \mid u \xrightarrow{a}\mid \cdot v \xrightarrow{a} v' \mid \\ &\leq \mid L \mid \end{aligned}$$

Where L is the total number of uncontrollable transitions of the community which is independent of the particular relation R and thus allow us to factor it out of the summation in the expression for complexity. Replacing the cost of *closure* and taking account that because the target is deterministic then $\mid v \xrightarrow{b} v' \mid \leq 1$ we get for the cost of a single call of $CONTROL(R)$ for a given R.

$$\begin{aligned} &\leq \mid L \mid \sum_{(s,t)\in R}\sum_{a\in\Sigma}\sum_{\alpha\in Com} \mid s \xrightarrow{\alpha|a}\mid \cdot \sum_{(u,v)\in R}\sum_{b\in\Sigma} \mid u \xrightarrow{\alpha|b} u' \mid \\ &\leq \mid L \mid \left(\sum_{(s,t)\in R}\sum_{a\in\Sigma}\sum_{\alpha\in Com} \mid s \xrightarrow{\alpha|a}\mid\right)^2 \\ &= \mid L \mid \cdot \mid L_R \mid^2 \end{aligned}$$

Where $\mid L_R \mid$ is the number of controllable transitions in R. Because the algorithm will visit every relation $R \in Z_0 = 2^{S\times S_t}$ at most once the total cost of a single run of the algorithm is therefore

$$\mid L \mid \cdot \sum_{R\in Z_0} \mid L_R \mid^2$$

The value of $|L_R|^2$ was computed in section 4.2.2 and we obtained

$$\begin{aligned}\sum_{R\in Z_0} |L_R|^2 &= N^2 \cdot \sum_{R\in Z_0} |R|^2 \\ &= N^2 \cdot (N + N^2)2^{N-2}\end{aligned}$$

Where with $N = |S \times S_t|$. The above is the cost of a single iteration of the algorithm which as we have seen will be rerun until the variable $change = false$. An upper bound for the number of the iterations of the algorithm is computed as follows. Referring to algorithm 11 we see that *change* is set to true iff a relation R is removed from Z *and* added to the set Y. Also relations are never removed from Y therefore the worst case occurs when initially $Y = \emptyset$ and at every iteration a single relation $R \in Z_0$ is added to Y until $Y = Z_0$ at which point the algorithm terminates. Therefore the number of iterations is at most $|Z_0|$. Finally, the total cost of the algorithm is

$$\begin{aligned}&|L| \cdot 2^N \cdot N^2 \cdot (N + N^2)2^{N-2} \\ &= O(|L| \cdot N^4 \cdot 2^N)\end{aligned}$$

Apart from the $|L|$ term the above worst case complexity is the same as the one obtained for the fixpoint algorithm in section 4.2.2.

4.4 Conclusion

In this chapter we developed methods to find a solution to the composition problem under partial information. Since it was shown in chapter 2 that an orchestrator exists if and only if a set of relations, called observation relations, exists the main focus of this chapter was to design algorithms for finding the set of observation relations for a given community and target service. Toward that end we developed two algorithms to compute a set of observation relations. One algorithm is a fixpoint algorithm that computes the largest possible set of observation relations. We showed that the algorithm is correct and its complexity is exponential in the size of the components and double exponential in the number of services. Also, we developed an on-the-fly algorithm to compute a set of observation relations for a given community and target. We also proved the correctness of the algorithm and the complexity was found to be similar to the fixpoint algorithm. Because of the elevated complexity of the problem, the on-the-fly algorithm is very promising since it visits states and sets of states in an incremental fashion. The advantage of the on-the-fly algorithm is that it does not need to compute a priori the whole belief state. It actually computes the belief states and seeks a solution at the same time. This means that it could find a solution or determine that no solution exists without computing the whole belief state. Since the computation of the whole belief state space is exponential in the number of services this is an important advantage of the on-the-fly algorithm over the fix-point approach.

Chapter 5

Conclusion and future work

5.1 Conclusion

In this thesis we have proposed a new model for the behavior composition problem and proposed efficient solutions.

In Chapter 2 we provided a framework for modeling the behavior composition problem. This was done by presenting a sufficiently rich model for the available services, the target service, the orchestrator, and the environment. The equivalence of behavior between the community under the partial control of the orchestrator, called the *orchestrated* community, and the target was defined. It was shown that this equivalence of behavior has a direct link to the classical problem of control theory. Our model extends the so called Roman Model in two ways. First, unlike the Roman Model, actions that are uncontrollable by the orchestrator are allowed in our model. Second, the target behavior can be specified using Modal specification which permits a much richer set of specifications, at no additional cost.

Also in Chapter 2, the existence of an orchestrator for the case of perfect and partial information was studied. First the existence of an orchestrator for the perfect information case was linked to the existence of a relation between the community and the target called a *controllability* relation. We proved that an orchestrator with perfect information exists if and only if a controllability relation between the community and the target exists. We proceeded to show how the orchestrator can be extracted from the controllability graph. Second the case when the orchestrator has partial information was studied. We proved that an orchestrator with partial information exists if and only if a set of relations, called *observation relations*, exists between the community and the target. The concept of *observation relations* is similar to the concept of *regions* in Petri nets. We also showed that the same results can be obtained if the target is specified using modal specification, and without any additional cost. Both cases, perfect and partial information, were studied in the presence of an environment that can impose constraints on the services.

After linking the existence of an orchestror to the existence of a *controllability* relation in the pefect information case, and a *set of observation relations* in the partial information case, we proposed algorithms for finding such relations.

In chapter 3 we proposed an on-the-fly algorithm to compute the *controllability* relation for the case of perfect information. The advantage of this algorithm is that, unlike fixpoint algorithms, it is not necessary that the algorithm visits the full state space of the community in order to obtain a solution, if one exists. Since the state space of the community is exponential in the number of services, this property could lead to significant improvement in practice. We proved that in the worst case the algorithm is polynomial in the size of a given service and exponential in the number of services, a complexity that matches the known lower bound. We also showed that our algorithm is robust with respect to service failure. If a service fails, the recomputation of the *controllability* relation does not start from scratch, but reuses information that were obtained before the failure. Also in Chapter 3 we proposed an *abstraction* method that reduces drastically the state space of the community and the target. We proved that if the *abstracted* community cannot simulate the *abstracted* target then no orchestrator exists, such

that the orchestrated community is a behavior composition of the target. Considering that the abstracted community could be up to 4 orders of magnitudes smaller that the original problem this results in significant speed up when no solution exits. Furthermore, when a simulation exists, it is used as a heuristic for the proposed on-the-fly algorithm to speed up the search for a solution. The proposed algorithm was shown to work with the Roman Model as well as the our extended model, including modal specification.

In chapter 4 we showed how the set of observation relations can be computed using a fixpoint algorithm. The complexity of the algorithm was studied and it was shown to be 2EXPTIME in the number of services. We also proposed an on-the-fly algorithm to compute the set of observation relations. We proved that the on-the-fly algorithm has the same complexity as the fixpoint algorithm. The importance of an on-the-fly algorithm for the partial information case cannot be over estimated. This algorithm has the expected advantage of an on-the-fly algorithm as in the perfect information case. Another big advantage of the proposed algorithm, is that most other methods compute the so called *belief state*, a sort of determinization procedure, before attempting to find a solution. The complexity of such a determinization procedure is exponential in the number of services which has to be done, *even if no solution exists*. By way of contrast, our proposed on the-fly-algorithm does the determinization on-the-fly and thus can determine quickly if a the problem has no solution without the need to compute the whole *belief state space*.

5.2 Future work

The work presented in this thesis can be extended in many directions. Below we discuss some of the possibilities.

5.2.1 Implementation of the algorithms

It is important to implement the proposed algorithms and use them on real world examples. The challenge is not the implementation but rather the translation of services into our model. For example translating BPEL documents into labelled transition systems with preconditions. In our model the services communicated via an environment. Even though intuitively one can see how this work, building an environment that can simulate the communication between services as specified in BPEL documents is a challenging task that will be attempted in the future.

5.2.2 State reduction of LTS

The abstraction technique we have proposed in chapter 3 is promising but it is somewhat ad-hoc. We plan on performing a more systematic study of *property preserving* reduction of labelled transition systems. One approach would be to use the rules proposed by Murata [Mur89] for Petri nets and adapt them to our model. Another approach would be to consider the target as logical formula and then use the standard abstraction and refinement technique used in model

checking [CGL94]. A complementary approach would be to use Ordered Binary Decision Diagrams (OBDD)[Bry92] to represent the services and the target. OBDDs provide an efficient representation of labelled transitions systems. We already did some work in this direction [FF12], but the main shortcoming is that OBDDs are used with fixpoint algorithms. Using OBDDs with on-the-fly algorithm has not been tackled before and it would be an important approach to pursue.

5.2.3 Partity games

There is a well known connection between parity games and controller synthesis [AVW03]. It would be interesting to explore the connection with behavior composition. There is already some work on the relation of behavior composition, in the special case of the Roman Model, with safety games such as [GPS13]. The classical approach to solving parity games, e.g. [Zie98][VJ00], involves what is called strategy iteration: starting with an initial strategy, these methods *improve* the current strategy until they find a wining strategy, if one exists. This iterative procedure is very similar to the fixpoint methods that we have discussed in this thesis. Recently, some promising on-the-fly algorithms to compute wining strategies for parity games were proposed [FL12]. In the future we plan to explore this connection further. In particular, we plan to investigate the potential of using the local search algorithms proposed in [FL12] to solve the behavior composition problem.

5.2.4 Quality of service and security

Most models of service composition, ours included, considers the services to be equivalent and therefore if two services can provide the same action the choice of the service is arbitrary. In many situations, however, the user or even the services have preferences. This is different from the "can/cannot perform action" paradigm that we have used. Given two services that can perform a certain action, it is possible that the user has a preference on which service should actually perform the action. This could be because of Quality of Service or for some security requirement. For example, a service prefers to communicate with another service only if it uses a specific authentication mechanism. In most approaches to QoS in service composition, e.g. [OAS+12], a given composition is obtained *then* its degree of preference is computed. Therefore any algorithm using such a procedure needs to compute all the solutions before deciding on the preferred solution. Incorporating QoS and security constraints in our proposed on-the-fly algorithms is a challenging task since these algorithms have "local" view of the composition and the "optimal" local choice from a QoS perspective does not necessarily lead to the "optimal" overall choice.

5.2.5 Distributed orchestration

One last important goal is to handle distributed orchestration. In reality not all the services belong to the same service provider. A useful approach for distributed orchestration would be to

divide the overall specification of the composition into sub-specifications that can be implemented by each provider. One possibility is to divide the specification based on the actions available from each service provider. Then each provider will have the task of synthesizing a sub-orchestrator for a sub-specification composed solely of actions that can be performed by the provider. Once all the sub-orchestrators are synthesized and collected an overall orchestrator is synthesized to combine the sub-orchestrator in such a way as to satisfy the overall specification.

The main challenge in this approach is to provide a procedure to split the overall specification according to some properties and give each service provider a goal for its own orchestration. The difficulty come from the necessity to split the overall specification in a way that guarantees that a meta-orchestrator exists whatever the implementation of every local specification is chosen. A possible solution may come from work on component-based design as in [RBB+11]. The computation of quotient formulae may prove to be a key concept for solving the distributed orchestration problem.

Bibliography

[ACKM04] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web services*. Springer Berlin Heidelberg, 2004.

[AVW03] A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *Theor. Comput. Sci.*, 303(1):7–34, 2003.

[BAHK10] Philippe Balbiani, Fahima Cheikh Alili, Pierre-Cyrille Héam, and Olga Kouchnarenko. Composition of services with constraints. *Electr. Notes Theor. Comput. Sci.*, 263:31–46, 2010.

[BCDG+05a] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Richard Hull, and Massimo Mecella. Automatic composition of transition-based semantic web services with messaging. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 613–624. VLDB Endowment, 2005.

[BCDG+05b] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Massimo Mecella. Automatic service composition based on behavioral descriptions. *International Journal of Cooperative Information Systems*, 14(04):333–376, 2005.

[BCF08a] P. Balbiani, F. Cheikh, and G. Feuillade. Composition of interactive web services based on controller synthesis. *Congress on Services - Part I, 2008. SERVICES '08. IEEE*, pages 521–528, July 2008.

[BCF08b] Philippe Balbiani, Fahima Cheikh, and Guillaume Feuillade. Composition of interactive web services based on controller synthesis. In *SERVICES I*, pages 521–528, 2008.

[BCF09] Philippe Balbiani, Fahima Cheikh, and Guillaume Feuillade. Algorithms and complexity of automata synthesis by asynhcronous orchestration with applications to web services composition. *Electronic Notes in Theoretical Computer Science*, 229(3):3 – 18, 2009. Proceedings of the First Interaction and Concurrency Experiences Workshop (ICE 2008).

[BCF10] Philippe Balbiani, Fahima Cheikh, and Guillaume Feuillade. Controller/orchestrator synthesis via filtration. *Electr. Notes Theor. Comput. Sci.*, 262:33–48, 2010.

[BCG+03] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Massimo Mecella. Automatic composition of e-services that export their behavior. In *ICSOC*, pages 43–58, 2003.

[BCGP08] Daniela Berardi, Fahima Cheikh, Giuseppe De Giacomo, and Fabio Patrizi. Automatic service composition via simulation. *Int. J. Found. Comput. Sci.*, 19(2):429–451, 2008.

[BCPT03] Piergiorgio Bertoli, Alessandro Cimatti, Marco Pistore, and Paolo Traverso. A framework for planning with extended goals under partial observability. In *ICAPS*, pages 215–225, 2003.

[BD98] Eric Badouel and Philippe Darondeau. Theory of regions. In *Lectures on Petri Nets I: Basic Models*, pages 529–586. Springer Berlin Heidelberg, 1998.

[BEL09] Sylvain Bouveret, Ulle Endriss, and Jérôme Lang. Conditional importance networks: A graphical language for representing ordinal, monotonic preferences over sets of goods. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pages 67–72, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[BPT06] Piergiorgio Bertoli, Marco Pistore, and Paolo Traverso. Automated web service composition by on-the-fly belief space search. In *ICAPS*, pages 358–361, 2006.

[BPT10] Piergiorgio Bertoli, Marco Pistore, and Paolo Traverso. Automated composition of web services via planning in asynchronous domains. *Artif. Intell.*, 174(3-4):316–361, March 2010.

[Bry92] Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24:293–318, September 1992.

[CGJ+03] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.

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

[CPRT03] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artif. Intell.*, 147(1-2):35–84, July 2003.

[CS01] R. Cleaveland and O. Sokolsky. *Handbook of Process Algebra*, chapter Equivalence and Preorder Checking for Finite-State Systems, pages 391–424. Elsevier, 2001.

[dAHK07] Luca de Alfaro, Thomas A. Henzinger, and Orna Kupferman. Concurrent reachability games. *Theor. Comput. Sci.*, 386(3):188–217, October 2007.

[DGF10] Giuseppe De Giacomo and Paolo Felli. Agent composition synthesis based on atl. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, pages 499–506, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[DGP10] Giuseppe De Giacomo and Fabio Patrizi. Automated composition of nondeterministic stateful services. In *Proceedings of the 6th international conference on Web services and formal methods*, WS-FM'09, pages 147–160, Berlin, Heidelberg, 2010. Springer-Verlag.

[DGS07] Giuseppe De Giacomo and Sebastian Sardina. Automatic synthesis of new behaviors from a library of available behaviors. In *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, pages 1866–1871, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[EHMR10] 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.

[FF12] H. Farhat and G. Feuillade. A symbolic method for the web service composition problem. In *Advances in Computational Tools for Engineering Applications (ACTEA), 2012 2nd International Conference on*, pages 182–185, Dec 2012.

[FF14] Hikmat Farhat and Guillaume Feuillade. Modal Specifications for Composition of Agent Behaviors. In *International Conference on Agents and Artificial Intelligence (ICAART)*, pages 437–444. SciTePress, mars 2014.

[FL12] Oliver Friedmann and Martin Lange. Two local strategy iteration schemes for parity game solving. *Int. J. Found. Comput. Sci.*, 23(3):669–685, 2012.

[FP07] Guillaume Feuillade and Sophie Pinchinat. Modal specifications for the control theory of discrete event systems. *Discrete Event Dynamic Systems*, 17(2):211–232, 2007.

[FVKR11] Yuzhang Feng, A. Veeramani, R. Kanagasabai, and Seungmin Rho. Automatic service composition via model checking. In *Services Computing Conference (APSCC), 2011 IEEE Asia-Pacific*, pages 477–482, 2011.

[GB96] Robert P. Goldman and Mark S. Boddy. Expressive planning and explicit knowledge. In *AIPS*, pages 110–117, 1996.

[GMP09] Giuseppe De Giacomo, Riccardo De Masellis, and Fabio Patrizi. Composition of partially observable services exporting their behaviour. In *ICAPS*, 2009.

[GPS13] Giuseppe De Giacomo, Fabio Patrizi, and Sebastian Sardiña. Automatic behavior composition synthesis. *Artif. Intell.*, 196:106–142, 2013.

[Koz83] Dexter Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.

[Mil71] Robin Milner. An algebraic definition of simulation between programs. In *Proceedings of the 2Nd International Joint Conference on Artificial Intelligence*, IJCAI'71, pages 481–489, San Francisco, CA, USA, 1971. Morgan Kaufmann Publishers Inc.

[Mur89] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr 1989.

[MW08] Anca Muscholl and Igor Walukiewicz. A lower bound on web services composition. *Logical Methods in Computer Science*, 4(2), 2008.

[oas07] OASIS WS-BPEL Technical Committee, Web Services Business Process Execution Language Version 2.0, 2007.

[OAS+12] ZacharyJ. Oster, SyedAdeel Ali, GaneshRam Santhanam, Samik Basu, and ParthaS. Roop. A service composition framework based on goal-oriented requirements engineering, model checking, and qualitative preference analysis. In Chengfei Liu, Heiko Ludwig, Farouk Toumani, and Qi Yu, editors, *Service-Oriented Computing*, volume 7636 of *Lecture Notes in Computer Science*, pages 283–297. Springer Berlin Heidelberg, 2012.

[OSB11] Zachary J. Oster, Ganesh Ram Santhanam, and Samik Basu. Identifying optimal composite services by decomposing the service composition problem. In *Proceedings of the 2011 IEEE International Conference on Web Services*, ICWS '11, pages 267–274, Washington, DC, USA, 2011. IEEE Computer Society.

[Par81] David Park. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, pages 167–183, London, UK, UK, 1981. Springer-Verlag.

[PBB+04] Marco Pistore, Fabio Barbon, Piergiorgio Bertoli, Dmitry Shaparau, and Paolo Traverso. Planning and monitoring web service composition. In *AIMSA*, pages 106–115, 2004.

[PF11] P. Papapanagiotou and J. Fleuriot. Formal verification of web services composition using linear logic and the pi-calculus. In *Web Services (ECOWS), 2011 Ninth IEEE European Conference on*, pages 31–38, 2011.

[PT87] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, December 1987.

[PT01] Marco Pistore and Paolo Traverso. Planning as model checking for extended goals in non-deterministic domains. In *IJCAI*, pages 479–486, 2001.

[RBB+11] J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, A. Legay, and R. Passerone. A modal interface theory for component-based design. *Fundamenta Informaticae*, 108(1-2):119–149, 2011.

[RS05] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In *Proceedings of the First international conference on Semantic Web Services and Web Process Composition*, SWSWPC'04, pages 43–54, Berlin, Heidelberg, 2005. Springer-Verlag.

[RW89] P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77:81–98, 1989.

[SP09] Thomas Ströder and Maurice Pagnucco. Realising deterministic behavior from multiple non-deterministic behaviors. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, pages 936–941, 2009.

[SPD08] Sebastian Sardina, Fabio Patrizi, and Giuseppe De Giacomo. Behavior composition in the presence of failure. In Gerhard Brewka and Jerome Lang, editors, *Proceedings of Principles of Knowledge Representation and Reasoning (KR)*, pages 640–650. AAAI Press, 2008.

[TP04] Paolo Traverso and Marco Pistore. Automated composition of semantic web services into executable processes. In *International Semantic Web Conference*, pages 380–394, 2004.

[vGW96] Rob J. van Glabbeek and W. Peter Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43:555–600, May 1996.

[VJ00] Jens Vöge and Marcin Jurdzinski. A discrete strategy improvement algorithm for solving parity games. In *CAV*, pages 202–215, 2000.

[WHH+06] Ralf Wimmer, Marc Herbstritt, Holger Hermanns, Kelley Strampp, and Bernd Becker. Sigref: a symbolic bisimulation tool box. In *Proceedings of the 4th international conference on Automated Technology for Verification and Analysis*, ATVA'06, pages 477–492. Springer-Verlag, 2006.

[Zie98] Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1–2):135 – 183, 1998.

[ZPG12] E. Zahoor, Olivier Perrin, and Claude Godart. Web services composition verification using satisfiability solving. In *Web Services (ICWS), 2012 IEEE 19th International Conference on*, pages 242–249, 2012.

Appendix

The goal of this appendix is to prove lemma 2.5.2 and theorem 2.5.3. The proof of lemma 2.5.2 is given first.

Proof of 2.5.2. Assume that a controllability relation R exists. We need to show that there exists an orchestrator with perfect information Ω such that for arbitrary $(t, e, s) \in R$ and $a \in \Sigma$ we have

$$\hat{\delta}_t(t, e, a) \neq \emptyset \Leftrightarrow \hat{\delta}_\Omega(s, e, a) \neq \emptyset$$

Furthermore, for all $t' \in \delta_t(t, e, a)$, $s' \in \delta_\Omega(s, e, a)$ and $e' \in \delta_E(e, a)$ we have $(t', e', s') \in R$.

Consider $(t, e, s) \in R$ and assume that $\hat{\delta}_t(t, e, a) \neq \emptyset$ then $t \xrightarrow{a} t' \wedge \delta_E(e, a) \neq \emptyset$. Since R is a controllability relation then

- Either $\delta_u(s, e, a) \neq \emptyset$ and for all $s' \in \delta_u(s, e, a)$, $e' \in \delta_E(e, a)$ we have $(t', e', s') \in R$. Recall that $\delta_\Omega(s, e, a) \supseteq \delta_u(s, e, a)$ thus $\delta_\Omega(s, e, a) \neq \emptyset$. Finally, $\hat{\delta}_\Omega(s, e, a) = \delta_\Omega(s, e, a) \times \delta_E(s, e) \neq \emptyset$.

- Or $\exists \alpha \in E(t, s), \delta_u(s, \alpha \mid a) \neq \emptyset$ and for all $s' \in \delta_u(s, a)$ and $e' \in \delta_E(e, a)$ we have $(t', e', s') \in R$. In this case choose $\Omega(s, t, \alpha) = 1$ then $\delta_\Omega(s, e, a) \supseteq \delta_u(s, e, \alpha \mid a) \odot \Omega(s, t, \alpha) \neq \emptyset$. Therefore, $\hat{\delta}_\Omega(s, e, a) = \delta_\Omega(s, e, a) \times \delta_E(s, e) \neq \emptyset$.

we have shown that $\hat{\delta}_t(t, e, a) \neq \emptyset \Rightarrow \hat{\delta}_\Omega(s, e, a) \neq \emptyset$.

Now assume that $\hat{\delta}_\Omega(s, e, a) = \delta_\Omega(s, e, a) \times \delta_E(s, a) \neq \emptyset$. Let τ be the sequence of actions executed to reach s from the initial state s^0. Since $\delta_\Omega = \delta_u(s, e, a) \bigcup_{\alpha \in Com} \delta_u(s, e, \alpha \mid a) \odot \Omega(s, \tau, \alpha)$ then there are two cases

- Either $\delta_u(s, e, a) \neq \emptyset$ then by definition of the controllability relation we get $\exists t'.t \xrightarrow{a} t'$ and $(t', e', s') \in R$ thus $\hat{\delta}_t = \delta_t(t, e, a) \times \delta_E(e, a) \neq \emptyset$. Furthermore, for all $s' \in \delta_u(s, e, a)$ and $e' \in \delta_E(e, a)$ we have $(t', e', s') \in R$.

- Or $\delta_u(s, \alpha \mid a) \neq \emptyset$ and $\Omega(s, \tau, \alpha) = 1$ for some $\alpha \in Com$. But from the construction in the first part we know that $\Omega(s, \tau, \alpha)$ is set to 1 only if $\alpha \in E(s, t)$ therefore by the definition of $E(s, t)$ we have $\exists t'.t \xrightarrow{a} t'$ and thus $\hat{\delta}_t(t, e, a) = \delta_t(t, e, a) \times \delta_E(e, a) \neq \emptyset$. Furthermore, for all $s' \in \delta_u(s, e, \alpha \mid a)$ and $e' \in \delta_E(e, a)$ we have $(t', e', s') \in R$.

We have shown that $\hat{\delta}_\Omega(s, e, a) \neq \emptyset \Rightarrow \hat{\delta}_t(t, e, a) \neq \emptyset$. Combining both results and the fact that $(t', e', s') \in R$ proves the lemma □

Proof of 2.5.3. ($\Rightarrow$) Assume that a controllability relation R exists. We need to show that there exists an orchestrator with perfect information Ω such that for all $\tau \in \Sigma^*$, $(s, e) \in \Delta_\Omega(s^0, e^0, \tau)$, $(t, e) \in \Delta(t^0, e^0, \tau)$ we have

$$\hat{\delta}_t(t, e, a) \Leftrightarrow \hat{\delta}_\Omega(s, e, a)$$

To do so we show a stronger version namely that in addition to the above we also have for all $(t', e') \in \Delta(t^0, e^0, \tau a)$ and for all $(s', e') \in \Delta(s^0, e^0, \tau a)$ we have $(t', e', s') \in R$.

The proof is by induction over the length of the trace.

Base case: The base case involves the empty string ϵ. We have $(t^0, e^0) \in \Delta_t(t^0, e^0, \epsilon)$ and $(s^0, e^0) \in \Delta_\Omega(s^0, e^0, \epsilon)$. Since $(t^0, e^0, s^0) \in R$ then from lemma 2.5.2 we know that for all $a \in \Sigma$ we have $\hat{\delta}_t(t^0, e^0, a) \neq \emptyset \Leftrightarrow \hat{\delta}_\Omega(s^0, e^0, a) \neq \emptyset$. Furthermore, for all$(t^1, e^1) \in \hat{\delta}_t(t^0, e^0, a)$ and $(s^1, e^1) \in \hat{\delta}_\Omega(s^0, e^0, a)$ we have $(t^1, e^1, s^1) \in R$.

Hypothesis: Assume that the above properties are true for the case $l-1$. Namely, given a trace τ of length $l-1$ then for all $(t^{l-1}, e^{l-1}) \in \Delta_t(t^0, e^0, \tau)$ and $(s^{l-1}, e^{l-1}) \in \Delta_\Omega(s^0, e^0, \tau)$ we have

$$\hat{\delta}_t(t^{l-1}, e^{l-1}, a) \neq \emptyset \Leftrightarrow \hat{\delta}_\Omega(s^{l-1}, e^{l-1}, a) \neq \emptyset$$

and for all $(t^l, e^l) \in \hat{\delta}_t(t^{l-1}, e^{l-1}, a)$, $(s^l, e^l) \in \hat{\delta}_\Omega(s^{l-1}, e^{l-1}, a)$ we have $(t^l, e^l, s^l) \in R$.

Induction step: Consider a trace of length l, τa and let $(t^l, e^l) \in \Delta_t(t^0, e^0, \tau a)$ and $(s^l, e^l) \in \Delta_\Omega(s^0, e^0, \tau a)$. From the definition of Δ we know that $(t^l, e^l) \in \hat{\delta}_t(t^{l-1}, e^{l-1}, a)$ and $(s^l, e^l) \in \hat{\delta}_\Omega(s^{l-1}, e^{l-1}, a)$ for some $(t^{l-1}, e^{l-1}) \in \Delta_t(t^0, e^0, \tau)$ and $(s^{l-1}, e^{l-1}) \in \Delta_\Omega(s^0, e^0, \tau)$. It follows from the hypothesis that $(t^t, e^l, s^l) \in R$ then by lemma 2.5.2 we finally get that for all $a \in \Sigma$:

$$\hat{\delta}_t(t^l, e^l, a) \neq \emptyset \Leftrightarrow \hat{\delta}_\Omega(s^l, e^l, a) \neq \emptyset$$

($\Leftarrow$) Assume that an orchestrator exists and define the relation

$$R = \{(t, e, s) \mid \exists \tau.(t, e) \in \Delta(t^0, e^0, \tau), (s, e) \in \Delta(s^0, e^0, \tau)\}$$

We show that R is a controllability relation.

Consider $(t, e, s) \in R$ then from the construction of R we have $(s, e) \in \Delta(s^0, e^0, \tau)$ and $(t, e) \in \Delta(t^0, e^0, \tau)$ for some $\tau \in \Sigma^*$. Suppose that $\hat{\delta}_t(t, e, a) = \delta_t(t, e, a) \times \delta_E(e, a) \neq \emptyset$. Since an orchestrator exists then $\hat{\delta}_\Omega(s, e, a) = \delta_\Omega(s, e, a) \times \delta_E(s, e, a) \neq \emptyset$. Because $\delta_E(e, a)$ is the same for community and target it follows that $\delta_t(t, e, a) \neq \emptyset \Rightarrow \delta_\Omega(s, e, a) \neq \emptyset$. Furthermore, from the definition of Δ we have for all $(t', e') \in \delta_t(t, e, a)$ and $(s', e') \in \delta_\Omega(s, e, a)$ that $(t', e') \in \Delta(t^0, e^0, \tau a)$ and $(s', e') \in \Delta(s^0, e^0, \tau a)$ thus $(t', e', s') \in R$. From the definition of $\delta_\Omega(s, e, a)$ and the fact that $\delta_\Omega(s, e, a) \neq \emptyset$ and for all $(t', e') \in \delta_t(t, e, a)$, $(s', e', a) \in \delta_\Omega(s, e, a)$ we have $(t', e', s') \in R$ we deduce that:

- Either $\delta_u(s, e, a) \neq \emptyset$ and for all $s' \in \delta_u(s, e, a)$ and $e' \in \delta_E(e, a)$ we have $(t', e', s') \in R$.
- Or $\delta_u(s, e, \alpha \mid a) \neq \emptyset$ and $\Omega(s, \tau, a) = 1$ for some $\alpha \in E(s, t)$ and and for all $s' \in \delta_u(s, e, a)$ and $e' \in \delta_E(e, a)$ we have $(t', e', s') \in R$.

Conversely, assume that $\hat{\delta}_\Omega(s, e, a) \neq \emptyset$ then $\delta_\Omega(s, e, a) \neq \emptyset$ and $\delta_E(e, a) \neq \emptyset$. Because an orchestrator exists we have $\hat{\delta}_t(t, e, a) \neq \emptyset$. It follows that $\delta_t(t, e, a) \neq \emptyset$. Furthermore, consider and arbitrary $(t', e') \in \hat{\delta}_t(t, e, a)$ and $(s', e') \in \delta_\Omega(s, e, a)$ then $(t', e') \in \Delta(t^0, e^0, \tau a)$ and $(s', e') \in \Delta(s^0, e^0, \tau a)$ whence $(t', e', s') \in R$ and it follows that R is an controllability relation according to definition 2.5.6. □

The final part of this appendix is a proof of theorem 2.5.3.

Proof of 2.5.3. ($\Rightarrow$). Suppose that a set of *observation relations*, Z, exists. We need to show that one can construct an orchestrator with partial information Ω such that for any arbitrary trace τ and for all $(t,e) \in \Delta(t^0, e^0, \tau)$, $(s,e) \in \Delta(s^0, e^0, \tau)$ we have the following:

$$\hat{\delta}_t(t, e, a) \neq \emptyset \Leftrightarrow \hat{\delta}_\Omega(s, e, a) \neq \emptyset$$

We proceed by induction on the length of τ.

Base case. By construction $(t^0, e^0, s^0) \in R$ for some $R \in Z$. Suppose that $\hat{\delta}_t(t^0, e^0, a) = \delta_t(t^0, e^0, a) \times \delta_E(e^0, a) \neq \emptyset$ then by the property of Z we have:

- Either $\delta_u(s^0, e^0, a) \neq \emptyset$ and for all $s' \in \delta_u(s^0, e^0, a)$,$t' \in \delta_t(t^0, e^0, a)$, $e' \in \delta_E(e^0, a)$ we have $(t', e', s') \in R$. But $\delta_\Omega(s^0, e^0, a) \supseteq \delta_u(s^0, e^0, a)$ thus $\hat{\delta}_\Omega(s^0, e^0, a) \neq \emptyset$.
- Or $\exists \alpha \in Com$ such that $\delta_u(s^0, e^0, \alpha \mid a) \neq \emptyset$ and for all $s' \in \delta_u(s^0, e^0, \alpha \mid a)$,$t' \in \delta_t(t^0, e^0, a)$ and $e' \in \delta_E(e^0, a)$, $(t', e', s') \in R'$ for some $R' \in Z$. Furthermore, for all $(u, v, w) \in R$ we have $w \xrightarrow{g(v), \alpha \mid b} w' \wedge v \xrightarrow{b} v' \Rightarrow \exists u'.u \xrightarrow{g(v) \mid b} u'$ with $(u', v', w') \in R'$. In this case we choose $\Omega(\sigma(s^0, \epsilon), \alpha) = 1$ then

  $$\delta_\Omega(s^0, e^0, a) \supseteq \delta_u(s^0, e^0, \alpha \mid a) \odot \Omega(\sigma(s^0, \epsilon), \alpha) \neq \emptyset$$

  Therefore $\hat{\delta}_\Omega(s^0, e^0, a) = \delta_\Omega(s^0, e^0, a) \times \delta_E(e^0, a) \neq \emptyset$

We have shown that $\hat{\delta}_t(t^0, e^0, a) \neq \emptyset \Rightarrow \hat{\delta}_\Omega(s^0, e^0, a) \neq \emptyset$.
Now we consider the opposite direction. Suppose that $\hat{\delta}_\Omega(s^0, e^0, a) = \delta_\Omega(s^0, e^0, a) \times \delta_E(e^0, a) \neq \emptyset$ then from the construction of $\delta_\Omega(s^0, e^0, a)$ there are two cases:

- Either $\delta_u(s^0, e^0, a) \neq \emptyset$ then by the properties of Z, $\delta_t(t^0, e^0, a) \neq \emptyset$ and thus $\hat{\delta}_t(t^0, e^0, a) \neq \emptyset$. Furthermore, for all $t' \in \delta_t(t^0, e^0, a)$,$s' \in \delta_u(s^0, e^0, a)$ and $e' \in \delta_E(e^0, a)$ we have $(t', e', t') \in R$.
- Or $\delta_u(s^0, e^0, \alpha \mid a) \neq \emptyset$ and $\Omega(\sigma(s^0, \epsilon), \alpha) = 1$ for some α. In the above construction Ω is set to 1 only if for all $(u, v, w) \in R$ we have $w \xrightarrow{g(v), \alpha \mid b} \wedge v \xrightarrow{b} v' \Rightarrow \exists u'.u \xrightarrow{g(v), b} u'$ with $(u', v', w') \in R'$ for some $R' \in Z$. In particular, $\exists t'.t^0 \xrightarrow{g(e^0), a} t'$ therefore $\hat{\delta}_t(t^0, e^0, a) \neq \emptyset$. Furthermore, $(t', e', s') \in R'$ for some $R' \in Z$.

We have shown that given a set of observation relations Z one can construct an orchestrator with partial information Ω such that $\hat{\delta}_t(t^0, e^0, a) \neq \emptyset \Leftrightarrow \hat{\delta}_\Omega(s^0, e^0, a) \neq \emptyset$. Moreover for all $e' \in \delta_E(e^0, a)$,$s' \in \delta_\Omega(s^0, e^0, a)$, $t' \in \delta_t(t^0, e^0, a)$ we have $(t', e', s') \in R$ for some $R \in Z$.

Induction hypothesis Assume that the above is true for traces τ of length $l-1$. This means that for all $(s^{l-1}, e^{l-1}) \in \Delta_\Omega(s^0, \tau)$, $(t^{l-1}, e^{l-1}) \in \Delta_t(t^0, \tau)$ and for all $a \in \Sigma$ we have $\hat{\delta}_t(t^{l-1}, e^{l-1}, a) \neq$

$\emptyset \Leftrightarrow \hat{\delta}_\Omega(s^{l-1}, e^{l-1}, a) \neq \emptyset$ and also for all $t^l \in \delta_t(t^{l-1}, e^{l-1}, a)$, $s^l \in \delta_\Omega(s^{l-1}, e^{l-1}, a)$, $e^l \in \delta_E(e^{l-1}, a)$ we have $(t^l, e^l, s^l) \in R$ for some $R \in Z$.

Induction step. Consider an arbitrary trace, τb of length l, with $(t^l, e^l) \in \Delta_t(t^0, e^0, \tau b)$, $(s^l, e^l) \in \Delta(s^0, e^0, \tau b)$. Suppose that for an arbitrary $a \in \Sigma$ we have $\hat{\delta}_t(t^l, e^l, a) = \delta_t(t^l, e^l, a) \times \delta_E(e^l, a) \neq \emptyset$. From the definition of Δ we know that $(t^l, e^l) \in \hat{\delta}_t(t^{l-1}, e^{l-1}, b)$, $(s^l, e^l) \in \hat{\delta}_\Omega(s^{l-1}, e^{l-1}, b)$ for some $(t^{l-1}, e^{l-1}) \in \Delta_t(t^0, e^0, \tau)$, $(s^{l-1}, e^{l-1} \in \Delta_\Omega(s^0, e^0, \tau)$. From the induction hypothesis this implies that $(t^l, e^l, s^l) \in R$ for some $R \in Z$. It follows from the properties of Z that

1. Either $\delta_u(s^l, e^l, a) \neq \emptyset$ and therefore $\hat{\delta}_\Omega(s^l, e^l, a) \neq \emptyset$. Furthermore, for all $s^{l+1} \in \delta_u(s^l, e^l, a)$,$t^{l+1} \in \delta_t(t^l, e^l, a)$,$e^{l+1} \in \delta_E(e^l, a)$ we have $(t^{l+1}, e^{l+1}, s^{l+1}) \in R$.
2. Or $\exists \alpha.\ \delta_u(s^l, e^l, \alpha \mid a) \neq \emptyset$ and $(t^{l+1}, e^{l+1}, s^{l+1}) \in R'$ for some $R' \in Z$. Then choose $\Omega(\sigma(s^l, \tau), \alpha) = 1$ hence $\delta_\Omega(s^l, e^l, a) \supseteq \delta_u(s^l, e^l, \alpha \mid a) \odot \Omega(\sigma(s^l, \tau), \alpha) \neq \emptyset$ and it follows that $\hat{\delta}_\Omega(s^l, e^l, a) \neq \emptyset$. Furthermore, α is chosen such that for all $(u, v, w) \in R$ we have $w \xrightarrow{g(v), \alpha \mid b} w' \wedge v \xrightarrow{b} v' \Rightarrow \exists u'.u \xrightarrow{g(v) \mid b} u'$ with $(u', v', w') \in R'$.

Now we check the reverse direction. Let $\hat{\delta}_\Omega(s^l, e^l, a) = \delta_\Omega(s^l, e^l, a) \times \delta_E(e^l, a) \neq \emptyset$. There are two cases:

1. Either $\exists \delta_u(s^l, e^l, a) \neq \emptyset$ then by the property of Z we have $\exists t^{l+1}.t^l \xrightarrow{g(e^l), a} t^{l+1}$ and it follows that $\hat{\delta}_t(t^l, e^l, a) \neq \emptyset$.
2. Or $\exists \alpha, \delta_u(s^l, e^l, a) \neq \emptyset$ and $\Omega(\sigma(s^l, \tau), \alpha) = 1$. By the way α is chosen we know that $\exists t^{l+1}.t^l \xrightarrow{g(e^l) \mid a} t^{l+1}$ and therefore $\hat{\delta}_t(t^l, e^l, a) \neq \emptyset$.

Therefore for all traces τ and for all $(s^l, e^l) \in \Delta(s^0, e^0, \tau)$ and $(t^l, e^l) \in \Delta(t^0, e^0, \tau)$ we have

$$\hat{\delta}_t(t^l, e^l, a) \neq \emptyset \Leftrightarrow \hat{\delta}_\Omega(s^l, e^l, a) \neq \emptyset$$

($\Leftarrow$). As before we use the concept of the observable set of the community but this time we add the environment to the mix. Let $Z = \{R_\theta \mid \theta \in \Theta\}$ be a set of sets of tuples defined as:

$$\begin{aligned} R_\theta = \{(t, e, s) \mid \forall x \in L(\theta) \exists \tau.(s, e) \in \Delta(s^0, e^0, \tau) \\ \wedge (t, e) \in \Delta(t^0, e^0, \tau) \\ \wedge \sigma(s, \tau) = x\} \end{aligned}$$

We need to show that if there exists an orchestrator with partial information, Ω, such that for all traces τ and for all $(s, e) \in \Delta_\Omega(s^0, e^0, \tau)$ and $(t, e) \in \Delta_t(t^0, e^0, \tau)$ if we have $\hat{\delta}_t(t, e, a) \neq \emptyset \Leftrightarrow \delta_\Omega(s, e, a) \neq \emptyset$ then Z is a set of observation relations.

Consider an arbitrary R_θ and choose an arbitrary tuple $(t, e, s) \in R_\theta$. Suppose that $(t', e') \in \hat{\delta}_t(t, e, a)$ then by assumption we have $\hat{\delta}_\Omega(s, e, a) \neq \emptyset$. Let $(s', e') \in \hat{\delta}_\Omega(s, e, a) = \delta_\Omega(s, e, a) \times \delta_E(e, a)$ then there are two cases:

1. Either $s' \in \delta_u(s, e, a)$ which means $s \xrightarrow{g(e),a} s'$. Since $(t, e, s) \in R_\theta$ then for all $x \in L(\theta),\ \exists \tau a$ such that

$$\begin{aligned}(s', e') \in \Delta_\Omega(s^0, e^0, \tau a) &\text{ because } (s, e) \in \Delta_\Omega(s^0, e^0, \tau)\\ &\text{and } s \xrightarrow{g(e),a} s'\\ &\text{and } e \xrightarrow{a} e'\\ (t', e') \in \Delta_t(s^0, e^0, \tau a) &\text{ because } (t, e) \in \Delta_t(t^0, e^0, \tau)\\ &\text{and } t \xrightarrow{g(e),a} t'\\ &\text{and } e \xrightarrow{a} e'\\ \sigma(s, \tau a) = x &\text{ because } \sigma(s, \tau) = x\\ &\text{and } s \xrightarrow{g(e),a} s'\end{aligned}$$

   therefore $(t', e', s') \in R_\theta$.

2. Or $\exists \alpha \in Com$ such that $s' \in \delta_u(s, e, \alpha \mid a) \neq \emptyset$ and $\Omega(\sigma(s, \tau), \alpha) = 1$. This means that $s \xrightarrow{g(e),\alpha|a} s'$. Then for all $x\alpha \in L(\theta\alpha),\ \exists \tau a$ such that

$$\begin{aligned}(s', e') \in \Delta_\Omega(s^0, e^0, \tau a) &\text{ because } (s, e) \in \Delta_\Omega(s^0, e^0, \tau)\\ &\text{and } s \xrightarrow{g(e),\alpha|a} s'\\ &\text{and } e \xrightarrow{a} e'\\ (t', e') \in \Delta_t(s^0, e^0, \tau a) &\text{ because } (t, e) \in \Delta_t(t^0, e^0, \tau)\\ &\text{and } t \xrightarrow{g(e),a} t'\\ &\text{and } e \xrightarrow{a} e'\\ \sigma(s, \tau a) = x\alpha &\text{ because } \sigma(s, \tau) = x\\ &\text{and } s \xrightarrow{g(e),\alpha|a} s'\end{aligned}$$

   Therefore $(t', e', s') \in R_{\theta\alpha} \in Z$.

   Furthermore, suppose that $(u, v, w) \in R_\theta$ and $w \xrightarrow{g(v),\alpha|b} w' \wedge v \xrightarrow{b} v'$ for some $b \in \Sigma$. From the definition of R_θ we know that for all $x \in L(\theta),\ \exists \lambda$ such that $(u, v) \in \Delta_t(t^0, e^0, \lambda)$, $(w, v) \in \Delta_\Omega(s^0, e^0, \lambda)$ and $\sigma(w, \lambda) = x$. Now since $w \xrightarrow{g(v),\alpha|b} w' \wedge v \xrightarrow{b} v'$ and $\Omega(w, x, \alpha) = 1$ then $\hat{\delta}_\Omega(w, v, b) \neq \emptyset$ which by assumption implies that $\hat{\delta}_t(u, v, b) \neq \emptyset$. Therefore for all

$x\alpha \in L(\theta\alpha),\ \exists \lambda b$ such that

$$
\begin{aligned}
(w', v') \in \Delta_\Omega(s^0, e^0, \lambda b) \text{ because } & (w, v) \in \Delta_\Omega(s^0, e^0, \lambda) \\
& \text{and } w \xrightarrow{g(v), \alpha | b} w' \\
& \text{and } v \xrightarrow{b} v' \\
(u', v') \in \Delta_t(t^0, e^0, \lambda b) \text{ because } & (u, v) \in \Delta_t(t^0, e^0, \lambda) \\
& \text{and } u \xrightarrow{g(e), b} u' \text{ by assumption} \\
& \text{and } v \xrightarrow{b} v' \\
\sigma(w', \lambda b) = x\alpha \text{ because } & \sigma(w, \lambda) = x \\
& \text{and } w \xrightarrow{g(v), \alpha | b} w'
\end{aligned}
$$

thus $(u', v', w') \in R_{\theta\alpha}$ and Z is a set of observation relations.

□

Résumé en Français

Chapitre 1: Introduction

Programmation orientée services

La programmation orientée service (Service Oriented Computing (SOC)) [ACKM04] est un paradigme de programmation qui supporte le développement rapide d'applications distribuées dans un environnement hétérogène en utilisant des composantes individuelles réutilisables, appelées des services. Ce fait peut être réalisé parce que les services peuvent être composés: les services peut être combinés de manière à produire un résultat, dont aucun des services individuels ne peut produire par lui-même.

Habituellement, il existe deux façons pour composer des services Web: par *orchestration* ou par *chorégraphie*. Dans cette thèse, nous étudions l'*orchestration*. Plus précisément, comment synthétiser un processus, appelé *orchestrateur*, de sorte que la communauté des services, contrôlées par l'orchestrateur, satisfait un objectif donné. Cette tâche de synthèse d'orchestration est appelée le problème de composition du comportement.

Le problème de composition du comportement a fait l'objet de recherche intensive. Ce fait peut être vu à partir des différentes approches pour le problème de la composition, allant de model checking [FVKR11], la planification des agents [DGS07], satisfiabilité [ZPG12], et la démonstration de théorèmes [PF11] (voir [RS05]). Le cadre que nous utilisons dans le présent document, est similaire à celui proposé en [BCG$^+$03], habituellement appelé le "Modèle Romain", et a été traité dans de nombreux ouvrages [BCF08a][DGP10][BCGP08].

Le problème de la composition est le suivant:

Étant donné un ensemble de services disponibles et une spécification du service but, est-ce que l'on peut synthétiser un orchestrateur qui combine les services d'une manière à répondre à la spécification.

Dans la plupart des travaux sur la composition des services Web, les services sont représentés par des machine à états finis (automates). Certaines approches utilisent des automates qui peuvent effectuer des actions de communication [OAS$^+$12], des actions internes [BCGP08], des actions internes non-observables [BPT10], des actions internes et de communication [BCDG$^+$05a] ou des actions de communication indirecte via un environnement [GPS13].

Notre approche

Notre approche consiste des composantes suivantes: un ensemble de services disponibles, un service but , un environnement, et un orchestrateur.

Les services disponibles, le service but, ainsi que l'environnement sont modélisés en tant que systèmes de transitions étiquetés. Le service but décrit un certain comportement que l'on doit satisfaire en utilisant les services disponibles. En d'autres termes, le service but joue le rôle d'une spécification. L'environnement représente tout ce qui n'est pas modelé par les services eux-mêmes. En d'autres termes, les services communiquent via l'environnement. Chaque service disponible peut effectuer un ensemble d'actions, certaines d'entre elles contrôlables et d'autres

sont incontrôlables. L'orchestrateur communique avec les services disponibles pour exécuter certaines actions. Le problème de la composition de comportement est défini comme suit:

Étant donné un ensemble de services disponibles, un service but et un environment, est-ce que l'on peut synthétiser un orchestrateur de telle sorte que le système composé des services disponibles contrôler par l'orchestrateur, en présence de l'environment, est équivalent au service but

Objectif et contributions de la thèse

Cette thèse aborde le problème de la composition de comportement. Même si nous positionnons notre travail dans le contexte des services Web, un comportement peut décrire la logique et l'interaction d'une composante, telles que des agents, en outre les services Web. En fait, une partie de ce travail a été utilisée pour résoudre le problème de comportement des multi-agents [FF14].

En abordant le problème de la composition, une préoccupation majeure a été le grand nombre d'états qui rendent les approches existantes trÃ¨s couteuses en terme de calculs. L'objectif de cette thèse est d'étendre l'expressivitÃ© des modÃ¨les et des spÃ©cifications tout en dÃ©veloppant des méthodes efficaces pour résoudre le problÃ¨me de composition de comportement. Nos contributions à cet égard sont:

1. Formuler un modèle qui prend en compte les transitions incontrôlables. Ce modèle est étudié dans le cas où l'orchestrateur a des informations *parfaites* ou *partielles*. Dans les deux cas, on a proposÃ© une condition nécessaire et suffisante pour l'existence d'une solution en termes d'une relation de contrôlabilité, dans le cas d'informations parfaites, et un ensemble de relations observables, dans le cas d'informations partielles. Cette caractérisation nous permet de concevoir des algorithmes efficaces pour la synthèse de l'orchestrateur. Nous avons montré comment un orchestrateur peut être synthétisé une fois que l'on a trouvé les relations.

2. Nous sommes allés au-delà des approches existantes en utilisant la *spécification modale*, qui est très expressive, pour modéliser le service but. Un service but modelisé en utilisant une spécification modale est essentiellement un *ensemble* de comportements acceptables.

3. Nous avons développé un algorithme à la volée pour le cas d'informations parfaites pour calculer la relation de contrôlabilité. L'importance d'un tel algorithme réside dans le fait que, à la différence d'autres approches, il n'a pas besoin de visiter tous les états du système qui sont généralement très nombreux. Un autre avantage qui n'est pas présent dans d'autres approches, est la possibilité d'utiliser des heuristiques pour réduire le nombre d'états visités. Nous avons également prouvé que notre algorithme proposé est robuste à la défaillance d'une composante, dans le sens que si une composante faillit il n'est pas nécessaire de redémarrer à partir de zéro, mais l'algorithme réutilise l'information recueillie avant l'échec.

4. Nous proposons une heuristique pour être utilisée avec l'algorithme mentionné ci-dessus, basé sur une méthode d'abstraction qui permet de réduire le nombres d'états de façon drastique. Une telle abstraction nous permet de déduire la non-existence d'une solution pour le problème initial de la non-existence d'une solution pour le problème abstrait, qui est plus petit. En outre, si une solution au problème abstrait existe, cette solution est utilisée comme une entrée à l'algorithme pour accélérer la recherche d'une solution au probléme original.

5. Nous avons développé un algorithme à la volée pour le cas d'informations partielles. Ici, le problème serait 2EXPTIME. En d'autres approches il est necessaire de déterminiser le système de transitions étiquetées avant le calcul d'une solution, même si aucune solution n'existe. Dans notre algorithme cette déterminisation est faite à la volée tout en trouvant une solution. Lorsqu'une solution n'existe pas cela accélérera considérablement les choses.

Organisation de la thèse

Dans le chapitre 2 nous présentons d'abord le modèle et les définitions de base. Nous montrons que notre modèle étend le Modèle Romain en deux façons. Premièrement, notre modèle inclus des actions incontrôlables qui sont absentes dans le Modèle Romain. Deuxièmement, nous utilisons des spécifications modales pour modéliser le service but. Les spécifications modales sont plus générales et moins restrictives pour la spécification du service but. Aussi dans le chapitre 2 nous prouvons qu'un orchestrateur avec informations parfaites existe si et seulement si une relation de *contrôlabilité* existe entre la communauté de services et le service but. Ce résultat est également prouvé pour le cas où la spécification modale est utilisée pour le but. De même, nous prouvons qu'un orchestrateur avec des informations partielles existe si et seulement si un ensemble de relations, que nous appelons *relations d'observation*, existe entre la communauté de services et le service but. Nous étendons aussi les résultats mentionnés ci-dessus au cas où un environnement est présent. Dans le chapitre 3, nous donnons un nouvel algorithme à la volée pour calculer la relation de contrôlabilité dans le cas des informations parfaites. On démontre que l'algorithme est correct et on calcule sa complexité. Nous avons également introduit une méthode d'abstraction qui est utilisée comme une heuristique pour accélérer l'algorithme. L'algorithme général qui comprend la spécification modale est également presenté. Dans le chapitre 4, le cas de l'orchestrateur avec des informations partielles est abordé. Nous présentons un algorithme pour calculer, à la volée, l'ensemble des *relations observation*. Cet algorithme permet d'éviter la construction de tous les sous-ensembles. Nous prouvons que l'algorithme est correct et on en calcule la complexité. Nous concluons et donnons les orientations futures dans le chapitre 5.

Chapitre 2: Résultats généraux

Ce chapitre contient les définitions nécessaires et le cadre formel du problème de la composition de comportement. Il contient également des résultats originaux qui font partie des contributions de cette thèse au problème de composition de comportement.

Un modèl de services

Dans cette thèse, les services sont modélisés comme des systèmes de transitions étiquetées (LTS). Les transitions d'un service donné peuvent être divisées en deux catégories: contrôlables et incontrôlables. Les premières peut être activées/ désactivées par un *orchestrateur* tandis que les secondes sont des transitions "spontanées" qui ne peuvent pas être contrôlées. Le systéme étudié est constitué de trois composantes:

1. Une communauté de n services disponibles.
2. Un service but dont le comportement nous devons imiter.
3. Un orchestrateur qui communique avec les services disponibles et qui a pour objectif de "réaliser" le comportement du service but.

Definition 5.2.1 (Service disponible)**.** *Un service disponible $\mathcal{S}_i$ est un tuple $\mathcal{S}_i = \langle S_i, \Sigma_i, Com_i, s_i^0, \delta_i \rangle$ où*

- *S_i est un ensemble fini d'états.*
- *$sigma_i$ est un alphabet finie d'action.*
- *Com_i est un ensemble fini de messages de communication.*
- *${}^s0_i$ est l'état initial.*
- *$\delta_i \subseteq S_i \times (\sigma_i \cup Com_i \times \sigma_i) \times S_i$ est la relation de transition.*

Le potentiel de composition de comportement réside dans le fait qu'il a de nombreux services ou composantes qui peuvent être orchestrés. Un tel ensemble de services est appelé dans cette thèse une *communauté de services disponibles*. C'est essentiellement le produit asynchrone de n services disponibles. La définition formelle est donnée par:

Definition 5.2.2 (Communauté des services)**.** *Une communautÃ© de n services disponibles $\mathcal{S}_i = \langle S_i, \sigma_i, Com_i, s_i^0, \delta_i \rangle$, $i = 1n \ldots$,*
est le tuple $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ où u

- *$S = S_1 \times \ldots \times S_n$.*
- *$s^0 = (s_1^0, \ldots s_n^0)$.*

- $\Sigma_u = \cup_i \Sigma_i$
- $Com = \cup_i Com_i$
- $\delta_u \subseteq S \times (\Sigma \cup Com \times \Sigma) \times S$

La relation de transition δ_u est le produit asynchrone de toutes les relations δ_i est défini comme:

$$(\langle s_1, \ldots, s_n \rangle, \alpha, \langle s'_1, \ldots, s'_n \rangle) \in \delta_u \text{ iff } (s_k, \alpha, s'_k) \in \delta_k \text{ for some } 1 \leq k \leq n \\ \text{and for all } i \neq k \text{ we have } s_i = s'_i$$

Definition 5.2.3 (Orchestrateur)**.** *Compte tenu d'une communauté de n services* $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$*, un orchestrateur pour cette communautÃ©© est une fonction* $\Omega : Com\ timesD \longrightarrow \{0, 1\}$*. Où D est un domaine de l'information qui à ce point n'est pas spécifié.*

Soit $\Omega(m)$, $m \in Com$, l'orchestrateur. Nous pouvons formaliser le comportement *orchestrée* (par Ω) de la communauté de service par l'introduction d'une fonction de transition *orchestré* δ_Ω défini par:

$$\delta_\Omega(s, a) = \delta_u(s, a) \bigcup_m \delta_u(s, m \mid a) \odot \Omega(m)$$

Notez que dans la définition ci-dessus le terme à gauche du symbole de union représente la contribution des transitions *incontrôlables* alors que le terme à la droite représente la contribution des transitions *contrôlables*. Le symbole $\odot$ est une notation pratique pour modéliser le on/off du comportement de l'orchestrateur. Cette opération peut être qualifiée par son effet:

$$\delta_u(s, m \,|\, a) \odot \Omega(m) = \begin{cases} \emptyset \text{ if } \Omega(m) = 0 \\ \delta_u(s, m \,|\, a) \text{ if } \Omega(m) = 1 \end{cases}$$

Maintenant, nous donnons la définition formelle du *service but*, que nous notons dans cette thèse par $\mathcal{S}_t$.

Definition 5.2.4 (Service but)**.** *Le service but* $\mathcal{S}_t$ *est le tuple* $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, \delta_t \rangle$ *où:*

- S_t *est un ensemble fini d'états.*
- Σ_t *est un alphabet finie d'action*
- t^0 *est l'état initial.*
- $\delta_t \subseteq S_t \times \Sigma \times S_t$ *est la relation de transition.*

Pour la communauté *orchestrée* , on défini la fonction $\Delta_\Omega : S \times \Sigma^* \to 2^S$, où $\Sigma = \Sigma_u \cup \Sigma_t$. Soit $a \in \Sigma$ et $x \in \Sigma^*$ des actions et sÃ©quence d'actions, respectivement, et $s \in S$ un état arbitraire de la *communauté* orchestrée , alors la fonction Δ est défini récursivement, avec l'aide de la séquence vide *epsilon*, comme:

$$\begin{aligned}\Delta_\Omega(s, \epsilon) &= \{s\} \\ \Delta_\Omega(s, xa) &= \bigcup_{s' \in \Delta_\Omega(s,x)} \delta_\Omega(s', a)\end{aligned}$$

On utilise le même symbole pour le service but

$$\begin{aligned}\Delta_t(t, \epsilon) &= \{t\} \\ \Delta_t(t, xa) &= \bigcup_{t' \in \Delta_t(s,x)} \delta_t(t', a)\end{aligned}$$

Composition avec informations parfaites

Definition 5.2.5 (Orchestrateur avec information parfaite)**.** *Compte tenu d'une communauté de nservices, $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$, un orchestrateur avec informations parfaites est une fonction $\Omega : S \times \Sigma^* \times Com \longrightarrow \{0, 1\}$.*

La définition de la fonction de transition *orchestrée* est alors:

$$\delta_\Omega(s, a) = \delta_u(s, a) \cup \left[\bigcup_{m \in Com} \delta_u(s, m \,|\, a) \odot \Omega(s, Tr(s), m) \right] \tag{5.1}$$

Où $Tr(s) \in \Sigma^*$ est la séquence d'actions réalisée par la communauté pour atteindre létat s.

Definition 5.2.6 (Communauté orchestrée)**.** *Soit $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ une communauté de n services, $\mathcal{S}_t = \langle S_T, \Sigma_t, t^0, \delta_t \rangle$ un service but et Ω un orchestrateur avec information parfaite. La communautÃ© orchestrÃ©e est dÃ©finie comme $\mathcal{S}_\Omega = \langle S, \Sigma, s^0, \delta_\Omega \rangle$.*

L'́volution de la *communauté* après une séquence d' actions est décrite par une extention de la relation de transition. Pour toute trace $x \in \Sigma^*$ et action $a \in \Sigma$ la fonction $\Delta_\Omega : S \times \Sigma^* \to 2^S$ est définie d'une manière récursive:

$$\begin{aligned}\Delta_\Omega(s, \epsilon) &= \{s\} \\ \Delta_\Omega(s, xa) &= \bigcup_{s \in \Delta_\Omega(s,x)} \delta_u(s', a) \cup \left[\bigcup_{m \in Com} \delta_u(s, m \,|\, a) \odot \Omega(s, x, m) \right]\end{aligned}$$

Composition de services

Le problème de la composition de comportement consiste à trouver un orchestrateur qui contrôle ou délègue des actions à différents services disponibles de sorte que la *communautée contrôlée* réalise (ou imite) le comportement d'un service but. Formellement,

Definition 5.2.7 (Composition de comportement avec information parfaite)**.** *Soit $\mathcal{S}_t$ un service but, mathcalS une communauté de n de services disponibles. Soit Ω un orchestrateur ave information parfaite et notons $\mathcal{S}_\Omega$ la communauté orchestré. Nous disons que $\mathcal{S}_\Omega$ est une composition de comportement de $\mathcal{S}_t$ ssi pour toutes les traces $\tau \in \Sigma^*$ et tout état du but $t \in \delta_t(t^0, \tau)$ et pour tout $s \in \Delta_\Omega(s^0, \tau)$ on a:*

$$\forall a \in \Sigma, \quad \delta_t(t, a) \neq \emptyset \leftrightarrow \delta_\Omega(s, a) \neq \emptyset$$

On peut caractériser l'orchestrateur en observant les états du service but. Cette correspondance nous permet de synthétiser un orchestrateur sans "se rappeler" de toutes les traces, plutôt en inspectant l'état du service but. Cette idée est utilisée en théorie du contrôle en utilisant le concept de contrôlabilité.

Controllability

Le concept d'une relation de contrôlabilité , ou contrôlabilité, est présenté ci-dessous.

Definition 5.2.8 (Controllabilité)**.** *Soit $\mathcal{S}_t = \langle S_t, \Sigma_t, s_t^0, \delta_t \rangle$ un service but et $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ une communautée de service. Une relation $R \subseteq S_t \times S$ est une controllability, si pour tous $(t, s) \in R$, $a \in \Sigma = \Sigma_u \cup \Sigma_t$ on a:*

1. *Si $\delta_u(s, a) \neq \emptyset$ alors $\exists t'.t \xrightarrow{a} t'$ et pour tous $s' \in \delta_u(s, a)$ on a $(t', s') \in R$.*
2. *$\exists E(s, t) \subseteq Com$ tel que pour tous $\alpha \in E(s, t)$ on a $s \xrightarrow{\alpha|a} s' \Rightarrow \exists t'.t \xrightarrow{a} t' \wedge (t', s') \in R$.*
3. *Si $t \xrightarrow{a} t'$ alors:*
   (a) *Soit $\delta_u(s, a) \neq \emptyset$ et pour tous $s' \in \delta_u(s, a)$ on a $(t', s') \in R$.*
   (b) *Ou $\exists \alpha \in E(s, t).\delta_u(s, \alpha \,|\, a) \neq \emptyset$ et pour tous $s' \in \delta_u(s, \alpha \,|\, a)$ on a $(t', s') \in R$*

Lemma 5.2.1. *Soit R une relation controllability entre le service but $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, \delta_t \rangle$ et la communautée $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$. Alors il existe un orchestrateur avec information parfaite, Ω, tel que pour tous $(t, s) \in R$, $t \in S_t$,$s \in S$, et pour tous $a \in \Sigma = \Sigma_u \cup \Sigma_t$ on a $\delta_t(t, a) \neq \emptyset \Leftrightarrow \delta_\Omega(s, a) \neq \emptyset$. De plus, pour tous $t' \in \delta_t(t, a)$ et $s' \in \delta_\Omega(s, a)$ on a $(t', s') \in R$.*

Le lemme ci-dessus nous aidera à demontré le théorème important qui caractérise l'existence d'un orchestrateur en termes de l'existence de la relation de contrôlabilité entre la communauté et le service but. Outre le modèle proposé, le théorème suivant est la première contribution de cette thèse au problème de composition de comportement.

Theorem 5.2.2. *Soit un service but $\mathcal{S}_t = \langle S_t, \Sigma_t, s_t^0, \delta_t \rangle$ et une communautée de n services disponibles $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$, alors un orchestrateur avec information parfaite, Ω, tel que $\mathcal{S}_\Omega$ est une composition de $\mathcal{S}_t$ ssi $\mathcal{S}$ est controllable par $\mathcal{S}_t$.*

Spécifications modales

Jusqu'à présent, le but de la composition de comportement était de satisfaire un *seul* comportement but. Nous pouvons étendre ce but en satisfaisant un *ensemble* de comportements but en utilisant les *spécifications modales*. Les spécifications modales ont été introduites pour modéliser des objectifs pour les problèmes de contrôle [FP07].

Definition 5.2.9 (spécification modale)**.** *Une spécification modale est un tuple $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, May, Must \rangle$ où*

- *S_t est un ensemble fini d'états.*
- *t^0 est l'état initiale.*
- *Σ_t est un ensemble fini d'actions.*
- *$May \subseteq S_t \times \Sigma \times S_t$ est la relation des transitions permis.*
- *$Must \subseteq May$ est la relation de transitions nécessaire.*

Après avoir défini la spécification modale on présente le problème de la composition.

Definition 5.2.10 (Spécification modale pour la composition de comportement)**.** *Soit $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ une communautée de services disponibles et $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, May, Must \rangle$ une spécification modale. Soit Ω un orchestrateur et on note par $S_\Omega = \langle S, \Sigma_u, s^0, \delta_\Omega \rangle$ la communautée orchestrée. On dit que S_Ω est une composition de comportement pour la spécification modale $\mathcal{S}_t$ ssi il existe une relation $\rho \subseteq S \times S_t$ tel que pour tous $(s,t) \in \rho$ et tous $a \in \Sigma = \Sigma_u \cup \Sigma_t$ on a:*

- $(t, a, t') \in Must \Rightarrow \delta_\Omega(s,a) \neq \emptyset \wedge \forall s' \in \delta_\Omega(s,a), (s', t') \in \rho$
- $\delta_\Omega(s,a) \neq \emptyset \Rightarrow \exists t'.(t, a, t') \in May \wedge \forall s' \in \delta_\Omega(s,a), (s', t') \in \rho$

La caractérisation de l'existence d'une composition dans le cas de la spécification modale est similaire aux cas précédents et dépend du concept de contrôlabilité.

Definition 5.2.11 (Controllabilité par rapport à une spécification modale)**.** *Soit $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ une communautée de services disponibles et $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, May, Must \rangle$ une spécification modale. On dit que $\mathcal{S}$ est controllable par rapport à $\mathcal{S}_t$ ssi il existe une relation $R \subseteq S \times S_t$ tel que $(s^0, t^0) \in R$ et pour tous $(s,t) \in R$ on a:*

- $s \xrightarrow{a} s' \Rightarrow \exists t'.(t, a, t') \in May \wedge (s', t') \in R$

- $\exists E(s,t) \subseteq Com$ *tel que tous* $\alpha \in E, b \in \Sigma$ *on a* $s \xrightarrow{\alpha|b} s' \Rightarrow \exists t'.(t,b,t') \in May \wedge (s',t') \in R$
- $(t,a,t') \in Must \Rightarrow$
  - *Soit* $\delta_u(s,a) \neq \emptyset$ *et pour tous* $s' \in \delta_u(s,a)$ *on a* $(s',t') \in R$
  - *Ou* $\exists \alpha \in E(s,t)$ *tel que* $\delta_u(s,\alpha\,|\,a) \neq \emptyset$.

Le théorème suivant est une généralisation des résultats précédents.

Theorem 5.2.3. *Soit* $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ *une communautée de service disponible et* $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, May, Must \rangle$ *une spécification modale. Un orchestrateur* Ω *existe tel que communitée orchestrée* $S_\Omega = \langle S, \Sigma_u, s^0, \delta_\Omega \rangle$ *est une composition de comportement pour la spécification modale* $\mathcal{S}_t$ *ssi* $\mathcal{S}$ *est controllable par rapport à* $\mathcal{S}_t$.

Orchestrateur avec informations partielles

Dans certains cas, les actions des services ne sont pas observables. Cela pourrait être le cas lorsque les services et l'orchestrateur sont de différents fournisseurs, alors l'orchestrateur n'a pas nécessairement accès aux services, sauf grâce à l'action de communication.

L'orchestrateur peut envoyer des messages à la communauté de services, à partir d'un ensemble *Com* et "se souvient" seulement des messages qu'il a déjà transmis. En d'autres termes, l'orchestrateur est supposé observer seulement ses propres transitions.

Definition 5.2.12 (Message trace)**.** *Une message trace est une fonction* $\sigma : S \times \Sigma^* \longrightarrow Com^*$ *qui associe avec chaque état* $s \in S$*, atteint par l'intermédiaire d'une séquence d'actions* $\tau \in \Sigma^*$*, la sequence des messages* $\sigma(s,\tau) \in Com^*$.

Dans ce cas la relation de transition devient:

$$\Delta_\Omega(s_0, \tau a) = \bigcup_{s \in \Delta(s^0,\tau)} \delta_\Omega(s,a)$$

où

$$\delta_\Omega(s,a) = \delta_u(s,a) \bigcup_{\alpha \in Com} \delta_u(s,\alpha\,|\,a) \odot \Omega\left(\sigma(s,\tau),\alpha\right)$$

et

$$\delta_u(s,\alpha\,|\,a) \odot \Omega(\sigma,\alpha) = \begin{cases} \emptyset & \text{If } \Omega(\sigma,\alpha) = 0 \\ \delta_u(s,\alpha\,|\,a) & \text{If } \Omega(\sigma,\alpha) = 1 \end{cases}$$

C'est la même formulation qu'avant sauf la dépendance de l'orchestrateur sur le message trace σ. Notez que la valeur de σ est dans Com^*.

Definition 5.2.13 (Composition de comportement avec information partielle)**.** *Soit $\mathcal{S}_t$ un service but et $\mathcal{S}$ une communautée de services. Soit Ω un orchestrateur avec information partielle et note par $\mathcal{S}_\Omega$ la communautée orchestré. On dit que $\mathcal{S}_\Omega$ est une composition de comportement de $\mathcal{S}_t$ avec information partielle ssi pour toutes traces $\tau \in \Sigma^*$ et pour tous états $t \in \Delta_t(t^0, \tau)$, $s \in \Delta_\Omega(s^0, \tau)$ on a:*

$$\forall a \in \Sigma, \quad \delta_t(t, a) \neq \emptyset \Leftrightarrow \delta_\Omega(s, a)$$

Encore une fois, à l'exception de σ, c'est la même définition que 5.2.7.

Comme dans le cas d'informations parfaites, l'existence d'un orchestrateur est caractérisée par l'existence d'un ensemble de relations au lieu d'une seule relation.

Relations d'observation

Avant d'introduire le concept de *relations d'observation* on donne quelques définitions nécessaires.

Definition 5.2.14 (State regular expression)**.** *Soit un état s, on associe une regular expression, θ_s, avec s defini par le language $L(\theta_s) = \{x \in Com^* \mid \exists \tau \in \Sigma^* \text{ with } \sigma(s, \tau) = x\}$.*

Definition 5.2.15 (Ensemble des observables)**.** *Soit Θ l'ensemble de tous les regular expressions de la communautée. L'ensemble des observables, est un ensemble $Z = \{R_\theta \mid \theta \in \Theta\}$ où chaque R_θ est defini par*

$$R_\theta = \{(t, s) \mid \forall x \in L(\theta) \exists \tau \in \Sigma^*. s \in \Delta(s^0, \tau) \wedge t \in \Delta(t^0, \tau) \\ \wedge \sigma(s, \tau) = x\}$$

Definition 5.2.16 (Relation d'observation)**.** *Un ensemble de relations $Z \subseteq 2^{S_t \times S}$ est dit relations d'observation ssi : pour chaque $R \in Z$ et pour tout $(t, s) \in R$ on a:*

1. *si $t \xrightarrow{a} t'$ alors*
   - *(a) Soit $\exists s'. s \xrightarrow{a} s' \wedge (t', s') \in R$*
   - *(b) Ou $\exists s', \alpha \in Com, R' \in Z$ avec*

$$s \xrightarrow{\alpha|a} s' \wedge (t', s') \in R' \\ \wedge \\ \forall (u, v) \in R (v \xrightarrow{\alpha|b} v' \Rightarrow \exists u'. u \xrightarrow{b} u' \wedge (u', v') \in R')$$

2. *si $s \xrightarrow{a} s'$ alors $\exists t'. t \xrightarrow{a} t' \wedge (t', s') \in R$*

Definition 5.2.17 (Controllabilité avec information partielle)**.** *Une communautée $\mathcal{S}$ est controllable avec information partielle par rapport a un service but $\mathcal{S}_t$ ssi il existe un ensemble de relations d'observation Z entre les états de $\mathcal{S}$ et $\mathcal{S}_t$ tel que $\exists R_0 \in Z$ with $(s^0, t^0) \in R_0$.*

La définition ci-dessus nous permet de caractériser l'existence d'un orchestrateur d'une manière similaire au cas des informations parfaites.

Theorem 5.2.4. *Soit $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ une communautée de services disponibles et $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, \delta_t \rangle$ un service but. Un orchestrateur avec information partielle, Ω, existe tel que $\mathcal{S}_\Omega$ est une composition de service de $\mathcal{S}_t$ ssi $\mathcal{S}$ est controllable avec information partielle par rapport a $\mathcal{S}_t$.*

Services communiquants

Jusqu'à présent, tous les services que nous avons examiné ne communiquent pas les uns avec les autres. La plupart des applications intéressantes de services, cependant, requièrent une certaine forme de communication ou au moins un mécanisme permettant le résultat d'une action de certains services d'être utilisé comme une entrée par un autre service. Nous avons étendu notre modèle avec un LTS supplémentaire appelé *environnement*. L'environnement interagit avec tous les services et leur permet ainsi de communiquer. Une action par un service peut changer l'état de l'environnement. Inversement, certaines actions de services peuvent dépendre de l'état de l'environnement. L'environnement non seulement représente une abstraction de la manière dont les données sont transférées, mais aussi il pourrait représenter le monde physique. Nous avons demontré que tous les résultats précédent, que nous avons présenté en l'absence de l'environnement sont également applicables en présence de l'environnement.

Chapitre 3: Orchestrateur avec informations parfaites

Dans ce chapitre, nous étudions le problème de la composition lorsque l'orchestrateur avec information parfaite. Ceci est fait en utilisant le fait qu'un orchestrateur existe si et seulement si une relation de contrôlabilité existe entre la communauté et le service but. Nous avons développé un algorithme efficace a la volée pour trouver une relation de contrôlabilité lorsqu'une existe.

Algorithme à la volée pour le Modèle Romain

Pour rendre la discussion simple, nous présentons d'abord l'algorithme pour le Modèle Romain, qui est un cas particulier de notre modèle. Dans une autre section, nous présentons l'algorithme pour nôtre modèle.

Soit S l'ensemble des états de la communauté des services disponibles, E l'ensemble des états de l'environnement et S_T l'ensemble des états du service but. L'algorithme maintient deux relations $\mathcal{A}$ et $\mathcal{B}$, les deux initialement vide.

La relation $\mathcal{A} \subseteq S_T \times E \times S$, représente la relation de contrôlabilité que l'algorithme essaie de trouver. Notez qu'il pourrait y avoir plus qu'une relation.

Pendant l'exécution de l'algorithme des tuples sont ajoutés et enlevés de la relation $\mathcal{A}$. $\mathcal{B} \subseteq S_T \times ES\times$ représente l'ensemble des tuples qui ont été trouvés par l'algorithme qu'ils n'appartient pas à la relation de contrôlabilité. A noter que les tuples sont ajoutés à $\mathcal{B}$ mais jamais retirées. L'ensemble $\mathcal{B}$ est maintenu de sorte qu'un tuple donné ne est pas traité plus d'une fois. L'algorithme se compose de deux fonctions mutuellement récursives $CONTROLE$ et $MATCH$.

Algorithm 14: Main routine for computing the controllability relation

```
1 MAIN
2 B ← ∅
3 while changed=true do
4 |   A ← ∅
5 |   changed=false
6 |   CONTROL(t^0, e^0, s^0)
7 return A
```

Notez que les tuples sont ajoutés à $\mathcal{A}$ par l'algorithme dans un (preorder traversal). Pour ça l'algorithme maintient une variable *changed* qui est mis Ã vrai à chaque fois un tuple est retiré de A. Si, quand l' algorithme termine $changed = true$, alors l'algorithme doit être exécuté à nouveau.

Notez que dès le début de chaque passe, la relation $\mathcal{A}$ est détruit (c'est à dire la valeur $\emptyset$) alors que la relation $\mathcal{B}$ qui maintient tout les tuples qui ne sont pas dans la relation de contrôlabilité

Algorithm 15: function CONTROL for the Roman Model case

```
1  CONTROL(t, e, s)
2  if ⟨t, e, s⟩ ∈ B then
3  |  return false
4  if ⟨t, e, s⟩ ∈ A then
5  |  return true
   /* Assume that (t,e,s) are in controllability relation                    */
6  A = A ∪ ⟨t, e, s⟩
7  res = true
   /* Check if controller can send messages to match all the transitions of
      the target                                                             */
8  foreach a ∈ Σ do
9  |  foreach t --g(e),a--> t' ∧ e --a--> e' do
10 |  |  res=MATCH (s, e --a--> e', t --a--> t')
11 |  |  if res = false then
12 |  |  |  Goto Exit
13 Exit:
14 if res=false then
15 |  B = B ∪ ⟨t, e, s⟩
16 |  A = A − ⟨t, e, s⟩
17 |  changed = true
18 return res
```

est maintenue d'une passe à l' autre.

Theorem 5.2.5. *L'algorithme CONTROLE se termine après un nombre fini d'étapes et lorsque il le fait retourne true ssi* (t^0, e^0, s^0) *est dans la relation de contrôlabilité.*

Theorem 5.2.6. *La complexité de l'algorithme CONTROL est polynomiale dans le nombre d'états d'un service donné et exponentielle dans le nombre de services.*

Robustesse par rapport a la faillite d'un service

Dans les systèmes complexes, il est possible pour certains des services disponibles de faillir. Cela pourrait être dû à la défaillance du service lui-même, son platform ou le canal de communication. Nous soutenons que l' algorithme proposée est robuste par rapport à l'échec d'un service. Nous montrons que le recalcul de la solution à l'aide de l'algorithme proposé n'a pas besoin de partir de zéro, mais peut utiliser des résultats déjaà obtenu avant la faillite du service pour accélérer le

Algorithm 16: function MATCH for the Roman Model case

```
1  MATCH(s, e --a--> e', t --a--> t')
   /* s_i is the i^th component of s,i.e.  s = ⟨s_1, ..., s_n⟩ similarly for s' and s''
      */
2  for i = 1 to n do
3  |  foreach s_i --g,a--> s'_i ∧ g_i(e) = true do
4  |  |  ENQUEUE (Q,i,s'_i)
5  res = false
6  while Q ≠ ∅ ∧ res = false do
7  |  s'_k = DEQUEUE(Q)
8  |  res=CONTROL(t', e', s')
9  |  if res=false then
10 |  |  Goto Label
11 |  else
12 |  |  foreach s_k --a--> s''_k do
13 |  |  |  res=CONTROL(t', e', s'')
14 |  |  |  if res = false then
15 |  |  |  |  Goto Label
16 |  Label:
17 return res
```

calcul. En particulier la relation $\mathcal{B}$ est maintenu. De cette manière, l'algorithme n'a pas à visiter les état dans $\mathcal{B}$.

L'idée principale se articule autour le théorème suivant.

Theorem 5.2.7. *Aprés que l'algorithme trouve une solution si* $(t, e, s) \in \mathcal{B}$ *et service l échoue, alors* $(t, e, s) \in \mathcal{B}$ *après l'échec.*

Comme le montre le théorème ci-dessus, l'algorithme que nous proposons est robuste dans le cas de l'échec. Si un service échoue après que la relation de contrôlabilité (et par conséquent l'orchestrateur) est calculé, le calcul après l'échec réutilise les informations obtenues avant l'échec.

Abstraction du problème de composition

L'un des avantages importants de l'algorithme proposé est la capacité de le combiner avec des heuristiques qui lui permettront de réduire l'espace de recherche et en consequant de réduire son coùt. Nous allons le faire en utilisant une technique d'abstraction qui réduit le nombre d'états du problème et rend donc la synthèse plus efficace.

L'abstraction proposé dans cette partie permet de déduire la non-existence d'un orchestrateur pour la communauté d'origine et le service but de la non-existence d'une relation de simulation entre la communauté abstraite et le service but abstrait. Plus important encore, quand une simulation existe entre la communauté abstraite et le service but abstrait , le résultat est utilisé en tant que *branch-and-bound* heuristique pour l'algorithme pour accélérer le calcul de la relation de contôlabilité, et donc la orchestrateur.

Le service but est abstrait en utilisant la branching bisimulation [vGW96]. La communauté est abstraite utilisant la *closure relation*.

Theorem 5.2.8. *Si aucune relation de simulation existe entre le service but abstrait et la communauté abstraite alors aucun orchestrateur Ω n'existe, telle que la communauté orchestré par Ω est une composition du service but.*

Heuristique pour la synthèse de l' orchestrateur

Nous construisons une heuristique basée sur l'abstraction. Pour ce faire, on suppose que $\bar{S}_t \prec \bar{S}$ et le résultat peut être utilisé comme entrée pour l'algorithme afin d'accélérer l'exécution. Supposons maintenant que $\bar{S}_t \prec \bar{S}$ et que la relation de simulation $R_{[]}$ entre les états de $\bar{S}$ et $\bar{S}_t$, a été déjà calculés. L'étape suivante consiste à incorporer les informations obtenu à partir de $R_{[]}$ dans l'algorithme. Ceci est réalisé en éliminant certaines transitions qui ne sont pas compatible avec l'information obtenue à partir de $R_{[]}$ ce qui permettra l'algorithme de visiter beaucoup moin d'états.

Cette idée simple mais puissante est incorporé dans le fonctionn $MATCH$ avec le résultat montré dans l'algorithme 17.

Algorithme pour le modèle général

Pour la simplicité de l'exposé nous avons jusqu'à prsent limités au cas particulier du Modèle Romain. Dans cette section, nous présentons l' extension de l'algorithme pour Ã notre modèle qui est plus général. Dans notre modèle l'orchestrateur (avec information parfaite) n'a pas de contrôlabilité complète sur la communauté. En d'autres termes, il ya des transitions de la communauté qui se produisent indépendamment de l'action du orchestrateur. Les autres transition peuevent être commandée par l' orchestrateur en envoyant des messages dans l' ensemble Com

Aussi, nous allons utiliser la spécification modale pour le service but. Nous utilisons des flèches en pointillés et solides pour les transitions *May* et *Must* respectivement. Si l'environnement est à l'état e et le service but est à l'état t nous écrivons $t \xrightarrow{g(e),a} t'$ pour $\exists g \in G.(t, g(e), a, t') \in May \wedge g(e) = true$ and $t \xrightarrow{g(e),a} t'$ for $\exists g \in G.(t, g(e), a, t') \in Must \wedge g(e) = true$.

De même à ce que nous avons fait dans précédemment pour le case du Modèle Romain, nous avont développé un algorithme qui utilise deux fonctions mutuellement récursive $CONTROLE$ et $MATCH$ qui sont représentés sur les algorithmes 18 and 19.

Algorithm 17: function MATCH when abstraction is used

```
1  MATCH(s, e -a-> e', t -a-> t')
   /* s_i is the i^th component of s,i.e.  s = ⟨s_1, ..., s_n⟩.  Similarly for s' and s'' */
2  for i = 1 to n do
3  |  foreach s_i -g(e),a-> s'_i do
   |  |  /* Determine the equivalence classes                                   */
4  |  |  x = D[t']
5  |  |  y = C[s']
   |  |  /* Consider a transition ONLY if the equivalence classes are similar */
6  |  |  if R[x][y] = 1 then
7  |  |  |  ENQUEUE (Q,i,s'_i)
8  res = false
9  while Q ≠ ∅ ∧ res = false do
10 |  s'_k = DEQUEUE(Q)
11 |  res=CONTROL(t', e', s')
12 |  if res=false then
13 |  |  Goto Label
14 |  else
15 |  |  foreach s_k -a-> s''_k do
16 |  |  |  res=CONTROL(t', e', s'')
17 |  |  |  if res = false then
18 |  |  |  |  Goto Label
19 |  Label:
20 return res
```

Algorithm 18: function CONTROL for the general model

```
1  CONTROL(t, e, s)
2  if ⟨t, e, s⟩ ∈ B then
3  |  return false
4  if ⟨t, e, s⟩ ∈ A then
5  |  return true
   /* Assume that (t,e,s) are in controllability relation                    */
6  A = A ∪ ⟨t, e, s⟩
7  res = true
8  Θ ← ∅
   /* Check that all uncontrolled community transitions can be matched by a
      May transition of the target                                           */
9  foreach a ∈ Σ do
10 |  foreach s --g(e),a--> s' ∧ e --a--> e' do
11 |  |  res = ∃t'.t --g(e),a--> t' ∧ CONTROL(t', e', s')
12 |  |  if res = false then
13 |  |  |  Goto Exit
14 |  |  Θ ← Θ ∪ {a}
   /* Check if orchestrator can send messages to match the remaining Must
      transitions of the target                                              */
15 foreach a ∈ Σ − Θ do
16 |  foreach t --g(e),a--> t' ∧ e --a--> e' do
17 |  |  res=MATCH (s, e --a--> e', t --a--> t')
18 |  |  if res = false then
19 |  |  |  Goto Exit
20 Exit:
21 if res=false then
22 |  B = B ∪ ⟨t, e, s⟩
23 |  A = A − ⟨t, e, s⟩
24 |  changed = true
25 return res
```

Algorithm 19: function MATCH for the general model

```
1 MATCH((s, e -a-> e', t -a-> t')
2 foreach α ∈ Com do
3     if s -g(e),α|a-> s' ∧ CONTROL(t', e', s') then
          /* Found a match by using α.  Now check it doesn't cause "side
             effects"                                                    */
4         foreach s -g(e),α|b-> s'' ∧ e -b-> e'' do
5             res = ∃t''.t -g(e),b-> t'' ∧ CONTROL(t'', e'', s'')
6             if res = false then /*this α does not work, try another one*/
7                 break;
8         return true
  /* No α matched                                                       */
9 return false
```

Chapitre 4: Orchestrateur avec informations partielles

Dans ce chapitre, le cas d'un orchestrateur avec observation partielle est abordée. Dans une telle situation, il existe des actions qui sont incontrôlable. De plus, l'orchestrateur ne sait pas exactement dans quel état est la communautée. Rappelons que nous avons prouvé au chapitre 2 qu'un orchestrateur avec informations partielles existe si et seulement si un ensemble de relations d'observations existe. Dans ce chapitre nous abordons deux contributions au problème de composition de comportement. D'abord, nous développons un algorithme point-fixe pour calculer l'ensemble des relations d'observation. Nous étudions également la complexité de l' algorithme. La deuxième contribution est un algorithme à la volée pour calculer l'ensemble des relations d'observations.

Definitions

Le concept d' observation est formalisé come suit. Soit $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ une communautée de services disponibles, et $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, \delta_t \rangle$ un service but don defini (see definition 2.4.1) la fonction *message trace*

$$\sigma : S \times \Sigma^* \longrightarrow Com^*$$

qui retourne la trace observable pour un état dans S et une trace dans Σ^*

Aussi la fonction de transition pour la communautée orchestré par un orchestrateur avec des informations partielles est donné par:

$$\delta_\Omega(s,a) = \delta_u(s,a) \bigcup_{\alpha \in Com} \delta_u(s, \alpha \mid a) \odot \Omega(\sigma(s,\tau), \alpha)$$

Où la trace $\tau \in \Sigma^*$ est determiné par l' history de s calculé par la fonction de transition qui est defini recursivement:

$$\Delta_\Omega(s^0, \epsilon) = \{s^0\}$$
$$\Delta_\Omega(s^0, \tau a) = \bigcup_{s \in \Delta_\Omega(s^0,\tau)} \left[\delta_u(s,a) \bigcup_{\alpha \in Com} \delta_u(s, \alpha \mid a) \odot \Omega(\sigma(s,\tau), \alpha) \right]$$

La composition doit satisfaire, pour toute trace τ et toute action a, si $s \in \Delta_\Omega(s^0, \tau)$ et $t \in \Delta_t(t^0, \tau)$ alors:

$$\delta_t(t,a) \neq \emptyset \Leftrightarrow \delta_\Omega(s,a) \neq \emptyset$$

On a deja demontrer(theorèm2.4.1), qu'un orchestrateur avec information partielle existe ssi il existe un ensemble de relations d' observation entre la communauté et le service but.

Algorithm 20: CLOSED returns true iff the input relation is closed with respect to uncontrollable actions

```
1 CLOSED(R)
2 foreach (s,t) ∈ R do
3 |   foreach s -a-> s' do
4 |   |   if t -a-> t' then
5 |   |   |   if (s',t') ∉ R then
6 |   |   |   |   return false
7 |   |   else
8 |   |   |   return false
9 return true
```

Algorithme point-fixe

L'algorithme point-fixe presenté dans cette section depend du concept de relation *fermée* qui est defini par:

Definition 5.2.18. *Soit $\mathcal{S} = \langle S, \Sigma_u, Com, s^0, \delta_u \rangle$ une communautée de services disponibles et $\mathcal{S}_t = \langle S_t, \Sigma_t, t^0, \delta_t \rangle$ le service but. Une relation $R \subseteq S \times S_t$ est dite fermée par rapport aux action uncontrôlables ssi:*

$$\forall (s,t) \in R, \forall a \in \Sigma_u : s \xrightarrow{a} s' \Rightarrow \exists t'.t \xrightarrow{a} t' \wedge (s',t') \in R$$

La définition ci-dessus prend essentiellement soins des transitions la incontrôlables. Ce qui reste est de synthétiser un orchestrateur pour contrôler les actions contrôlables. À cette fin, nous utilisont un algorithme point-fixe similaire au cas d'information parfaite. Cet algorithme appartient également à la catégorie que nous avons appelé "point fixe" parce que le point de départ est tout l'ensemble de tous les sous-ensembles de $S \times S_T$.

$$Z_0 = 2^{S \times S_t}$$

Aprés, l'ensemble de toutes les relations fermés dans Z_0 est calculé:

$$Z_1 = \{R \in Z_0 \mid CLOSED(R)\}$$

Le calcul de l'ensemble Z_1 est représentée dans l'algorithme 21. Les accolades dans la ligne 4 de l'algorithme sont utilisé pour insister sur le fait qu'une relation fermée R est un *des élément* de Z_1. Soit la fonction suivante:

Algorithm 21: Computing Z_1

```
1 Z_1 ← ∅
2 foreach R ⊆ S × S_t do
3 |   if CLOSED (R) then
4 |   |   Z_1 ← Z_1 ∪ {R}
5 return Z_1
```

$$
\begin{aligned}
F(Z_i) = \{R \in Z_i \mid \forall (s,t) \in R : \\
& t \xrightarrow{a} t' \Rightarrow \left(\exists s'.s \xrightarrow{a} s' \wedge (s',t') \in R\right) && \text{(5.2a)} \\
& \vee \\
& \Big(\exists \alpha \in Com, s' \in S, R' \in Z_i.s \xrightarrow{\alpha|a} s' \wedge (s',t') \in R' && \text{(5.2b)} \\
& \qquad \wedge \\
& \quad \forall (u,v) \in R : u \xrightarrow{\alpha|b} u' \Rightarrow v \xrightarrow{b} v' \wedge (u',v') \in R' \quad \Big) && \text{(5.2c)}
\end{aligned}
$$

Pour calculer l'ensemble des relations d'observations l'algorithme applique plusieurs foit la fonction F commenąant par l'ensemble des relations Z_1. Par conséquent, l'algorithme calcule la séquence:

$$Z_{i+1} = F(Z_i)$$

la relation Z_0 et donc Z_1 est fini, alors le procédure ci-dessus se termine après un nombre fini d'étapes, soit j, defini par $F(Z_j) = Z_j$. L'ensemble des relations Z_j a, par la construction de Z_1 et F, les propriétés d'un ensemble de relations d'observations.

Theorem 5.2.9. *Soit $N = \mid S \times S_t \mid$ alors la complexité de l'algorithm point-fixe est*

$$O\left(N^4 \cdot 2^{2N}\right)$$

Un algorithme à la volée

Dans cette section, nous présentons un algorithme pour construire progressivement les relations d'observations. Cet approche incrémentale est plus efficace que le calcul de toutes les sous-ensembles.

Algorithm 22: On the fly algorithm

```
1  clos ← CLOSURE((∅, s^0, t^0))
2  if clos = ∅ then
3  |  return ∅
4
5  changed ← true
6  while changed = true do
7  |  changed ← false
8  |  Z ← ∅
9  |  res = CONTROL(clos)
10 |  if res = false then
11 |  |  break
12 return Z
```

Le but de l'algorithme à la volée présenté dans cette section est de calculer un ensemble de relations d'observations, Z. Une condition nécessaire pour une relation R d'appartenir à Z est que R soit fermé. Nous définissons un procédure qui calcule la fermeture d'un tuple. Cette opération de fermeture est effectuée le procédure $CLOSURE$ qui est défini d'une manière inductive et présenté dans l'algorithme 23.

La fonction $CONTROL(R)$ retourne vrai si la relation R appartient à l'ensemble cherché des relations Z, sinon elle retourne faux. Le fonctionnement generale de l'algorithme est représenté dans l'algorithme 22.

La fonction $CONTROL$ prend une relation R en tant que paramètre et retourne vrai si pour chaque (s, t) dans R et pour chaque transition du service but on peut trouvé une transition correspondante de la communautée. Cette correspdondance est effectuée par la fonction $MATCH$. Le procédure $MATCH$, utilise un procédure auxiliaire $CHECK$

Exactitude et complexité

Theorem 5.2.10. *Let L be the total number of uncontrollable transitions of the community and $N = \mid S \times S_t \mid$ the on-the-fly algorithm computes the set of observation relations and its complexity is*

$$\begin{aligned} &\mid L \mid \cdot 2^N \cdot N^2 \cdot (N + N^2) 2^{N-2} \\ &= O(\mid L \mid \cdot N^4 \cdot 2^N) \end{aligned}$$

Algorithm 23: Function CLOSURE

```
1  CLOSURE(R,(s,t))
2  if (s,t) ∈ bad then
3  |  return ∅
4  if (s,t) ∈ R then
5  |  return R
6  R' ← R ∪ {(s,t)}
7  foreach s -a-> s' do
8  |  if t -a-> t' then
9  |  |  R' ← CLOSURE(R', (s',t'))
10 |  |  if R' = ∅ then
11 |  |  |  bad ← bad ∪ {(s',t')}
12 |  |  |  return ∅
13 |  else
14 |  |  bad ← bad ∪ {(s,t)}
15 |  |  return ∅
16 return R'
```

Algorithm 24: Function CONTROL

```
1  CONTROL(R)
2  if R ∈ Y then
3  |  return false
4  if R ∈ Z then
5  |  return true
6  Z ← Z ∪ {R}
7  foreach (s,t) ∈ R do
8  |  foreach t -a-> t' do
9  |  |  res = MATCH (R,s,t -a-> t')
10 |  |  if res = false then
11 |  |  |  GOTO Exit
12 return true
13 Exit: Z ← Z − R
14 Y ← Y ∪ R
15 changed ← true
16 return false
```

Algorithm 25: Function MATCH

```
1  MATCH (R, s, t --a--> t')
2  if s --a--> s' then
3  |  return true
4  foreach α ∈ Com do
5  |  if s --α|a--> s' then
   |  |  /* check to see if this α leads to a closed set                */
6  |  |  R' ←CHECK (R, α)
7  |  |  if R' = ∅ then
   |  |  |  /* try different α this one leads to non-closed set         */
8  |  |  |  Continue
9  |  |  else
10 |  |  |  res =CONTROL (R')
11 |  |  |  if res = true then
12 |  |  |  |  return true
   /* tried all α's.  none worked                                        */
13 return false
```

Algorithm 26: Function CHECK

```
1  CHECK (R, α)
2  R' ← ∅
3  foreach (u, v) ∈ R do
4  |  if u --α|b--> u' then
5  |  |  if v --b--> v' then
6  |  |  |  clos = CLOSURE(∅, u', v')
7  |  |  |  if clos = ∅ then
8  |  |  |  |  return ∅
9  |  |  |  else
10 |  |  |  |  R' ← R' ∪ clos
11 |  |  else
12 |  |  |  return ∅
13 |  |  x
14 return R'
```

Chapitre 5:Conclusion

Dans cette thèse, nous avons proposé un nouveau modèle et des solutions efficaces pour le problème de composition de comportement.

Dans chapitre 2 nous avons fourni un cadre pour la modélisation du problème de composition de comportement. Cela a été fait en présentant un modèle suffisamment riche pour les services disponibles, le service but, l'orchestrateur, et l'environnement. L'équivalence des comportement entre la communauté sous le contrôle partiel par l' orchestrateur, et le service but était défini. On a montré que cette équivalence de comportement a un lien direct avec le problème classique de la théorie du contrôle. Notre modèle étend le Modèle Romain de deux façons. Premièrement, contrairement au Modèle Romain, nôtre modèle inclus des actions qui sont incontrôlables par l'orchestrateur. Deuxièmement, le comportement du service but peut être spécifié en utilisant les spécifications modales ce qui permet des spécifications plus expressives, sans coût supplémentaire.

Toujours dans chapitre 2, l'existence d'un orchestrateur pour le cas des informations parfaites et partielles a été étudié. Tout d'abord l'existence d'un orchestrateur pour le cas des informations parfaite a été lié à l'existence d'une relation entre la communauté et le service but appelé *contrôlabilité*. Nous avons prouvé qu'un orchestrateur avec des informations parfaite existe si et seulement si une relations de *contrôlabilité* existe entre la communauté et le service but. Nous avons aussi montré comment l' orchestrateur peut être synthesisé à partir de la relation de *contrôlabilité*. Le cas où l'orchestrateur a des information partielle a été aussi étudié. Nous avons prouvé qu'un orchestrateur avec des informations partielles existe si et seulement si un ensemble de relations, appelé *relations d'observation*, existe entre la communauté et le service but. Le concept de *relations d'observation* est similaire à la notion de *régions* pour les réseaux de Petri. Nous avons également montré que les mêmes résultats peuvent être obtenu si on utilise les spécifications modales pour le service but, et sans aucun coût supplémentaire.

Nous avons aussi proposé des algorithmes pour trouver la relation de *contrôlabilité*, pour le cas des information parfaites, et les *relations d'observations* dans le cas des informations partielles.

Dans chapitre 3 nous avons proposé un algorithme à la volée pour calculer la relations de *contrôlabilité* pour le cas des informations parfaites. L' avantage de cet algorithme est que, contrairement à des algorithmes de point-fixe, il n'est pas nécessaire que l'algorithme visite tout l'espace d 'état de la communauté afin d'obtenir une solution. Étant donné que l'espace d'état de la communauté est exponentiel en nombre de services, cette propriété abouti à une amélioration significative dans la pratique. Nous avons prouvé que, dans le pire des cas l'algorithme est polynomial par rapport à la taille d'un service donné et exponentielle par rapport au nombre de services, une complexité qui correspond à la limite inférieure connu. Nous avons également montré que notre algorithme est robuste par rapport à l'échec d'un service. Si un service échoue, le calcul de la relation de *contrôlabilité* ne démarre pas à partir de zéro, mais réutilise des informations qui ont été obtenues avant l'échec. Toujours dans chapitre 3 nous avons proposé une *méthode d'abstraction* qui réduit considérablement l'espace d'état de la communauté. Nous avons prouvé que si la communauté *abstraite* ne simule pas le service but abstrait alors un orchestrateur n'existe pas. En outre, quand une simulation existe, elle est utilisé comme une

heuristique pour l'algorithme à la volée pour accélérer la recherche pour une solution.

Dans chapitre 4 nous avons montré comment les relations d'observation peuvent être calculés en utilisant un algorithme de point-fixe. La complexité de l'algorithme a été étudié et on a démontré qu'il est 2EXPTIME par rapport au nombre de services. Nous avons également proposé un algorithme à la volée pour calculer les relations d'observation. Nous avons prouvé que l'algorithme à la volée a la même complexité que l'algorithme point-fixe. L'importance d'un algorithme à la volée pour le cas des informations partielles est que la plupart des autres méthodes doit calculer les *belief states*, une sorte de procédure de déterminisation, avant de tenter de trouver une solution. La complexité d'une tel procédure de déterminisation est exponentielle par rapport au nombre de services qui doit être fait, *même si aucune solution n'existe pas.* Par contraste, notre algorithme à la volée fait la déterminisation à la volée et peut ainsi déterminer rapidement si un problème n'a pas une solution sans pre-calculer les *belief states.*

Travaux futurs

Le travail présenté dans cette thèse peut être étendu dans de nombreux directions. Ci-dessous nous discutons certaines possibilités.

Implementation des algorithmes

Il est important d'implementer les algorithmes proposés et les utiliser avec des exemples pratique. Le défi est plutôt de transformer les services en automates . Par exemple la traduction de documents BPEL dans les systèmes de transitions étiquetées avec conditions préalables. Dans notre modèle, les services communiquent via un environnement. Même si intuitivement on peut imaginer le principe général, créer un environnement qui permet de simuler la communication entre services comme spécifié dans les documents BPEL est une tâche difficile qui sera tentée à l'avenir.

Reduction d'états

La technique d'abstraction que nous avons proposée dans chapitre 3 est prometteuse, mais elle est un peu ad-hoc. Nous prévoyons effectuer une étude plus systématique en utilisant la reduction *property preserving* pour les systèmes de transitions étiquetées. Un approche serait d'utiliser les règles proposé par Murata [Mur89] pour les réseaux de Petri et les adapter à notre modèle. Un autre approche serait de considérer le service but comme formule logique et ensuite utiliser l'abstraction et le raffinement technique utilisée dans la vérification de modèle [CGL94]. Une approche complémentaire serait d'utiliser les (OBDD) [Bry92] pour représenter les services. Les OBDDs fournissent une représentation efficace des systèmes a transitions étiquetées. Nous avons déjà fait quelques travaux dans cette direction [FF12], mais le principal défaut est que les OBDDs sont utilisés avec des algorithmes de point-fixe. Utilisation OBDDs avec des algorithmes à la volée n'a pas été abordée auparavant et il serait une piste importante pour poursuivre.

Jeux de parité

Il ya un lien bien connu entre les jeux de parité et la synthèse d'un contrôleur [AVW03]. Il serait intéressant d'explorer la connexion avec la composition de comportement. Il ya déjà un certain travail sur la relation de la composition de comportement, dans le cas particulier du Modèle Romain, avec les jeux de sécurité (Safety games) tels que [GPS13]. Les approches classiques pour trouver une solution aux jeux de parité, par exemple [Zie98] [VJ00], utilisent ce qu'on appelle la stratégie d'itération: commençant par une stratégie initiale, ces méthodes *améliorent* la stratégie actuelle jusqu'à ce qu'ils trouvent une stratégie gagnante. Cette procédure itérative est très similaire aux méthodes de point-fixe que nous avons discuté dans cette thèse. Récemment, certains algorithmes prometteurs pour calculer des stratégies gagnantes, à la volée, pour les jeux de parité ont été proposées [FL12]. Dans l'avenir, nous prévoyons explorer cette connexion plus loin. En particulier, nous prévoyons d'étudier le potentiel de en utilisant les algorithmes de recherche locale proposées dans [FL12] pour résoudre le problème de composition de comportement.

Qualité de service (QoS) et sécurité

La plupart des modèles de composition de services, y compris le nôtre, considèrent les services comme équivalents et donc si deux services peuvent fournir la même action le choix du service est arbitraire. Dans de nombreuses situations, cependant, l'utilisateur ou même les services ont des préférences. Étant donné que deux services peuvent effectuer une certaine action, il est possible que l'utilisateur ait une préférence pour le service devrait en fait exécuter l'action. Cela pourrait être en raison de la qualité de service ou pour une exigence de sécurité. Par exemple, un service préfère communiquer avec un autre service s'il utilise un mécanisme spécifique d'authentification. Dans la plupart des approches de QoS pour la composition de service, par exemple [OAS+12], on obtient une composition donnée puis son degré de préférence est calculée. Par conséquent, tout algorithme utilisant une telle procédure doit calculer toutes les solutions avant de décider la solution préférée. Intégrer la QoS et les contraintes de sécurité dans nos algorithmes à la volée est une tâche difficile car ces algorithmes ont une vue "locale" de la composition et le choix "optimal" du point de vue local ne conduit pas nécessairement à un choix "optimal" global .

Distributed orchestration

Un dernier objectif important est de gérer l'orchestration distribué. En pratique, tous les services n'appartient pas au même fournisseur de services. Un approche utile pour une orchestration distribuée serait de diviser la spécification globale de la composition en sous-spécifications. Une possibilité consiste à diviser la spécification basée sur les actions disponibles de chaque service fournisseur. Puis chaque fournisseur aura la tâche de synthétiser un sous-orchestrateur pour une sous-spécification composé uniquement d'actions qui peuvent être effectuées par le fournisseur. Une fois que tous les sous-orchestrateurs sont synthétisés, un orchestrateur global est synthétisé en combinant les sous-orchestrateurs de manière à satisfaire la spécification globale.

Le principal défi de cette approche est de fournir une procédure pour diviser la spécification globale selon certaines propriétés et donner à chaque fournisseur de services un but pour sa propre orchestration. La difficulté viendra de la nécessité de diviser la spécification globale d'une manière qui garantit que un méta-orchestrateur existe. Une solution peut provenir des approche (component-based design) [RBB+11]. Le calcul de formules de quotient peut s'avérer être un concept clé pour résoudre le problème d'orchestration distribué.