COMPOSITION DE COMPOSANTS DYNAMIQUES BASÉE SUR DES DESCRIPTIONS DE LEUR COMPORTEMENT

COMPOSITION OF DYNAMIC COMPONENTS BASED ON BEHAVIORAL DESCRIPTIONS

par

Masoud Barati

thèse présentée au Département d'informatique en vue
de l'obtention du grade de docteur ès sciences (Ph.D.)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, May 18, 2018

Le 18 mai 2018

le jury a accepté la thèse de Monsieur Masoud Barati
dans sa version finale.

Membres du jury

Professeur Richard St-Denis
Directeur de recherche
Département d'informatique
Université de Sherbrooke

Professeur Luc Lavoie
Président rapporteur
Département d'informatique
Université de Sherbrooke

Professeur Marc Frappier
Évaluateur interne au programme
Département d'informatique
Université de Sherbrooke

Professeur Froduald Kabanza
Évaluateur interne au programme
Département d'informatique
Université de Sherbrooke

Professeur Stefan D. Bruda
Évaluateur externe au programme
Department of Computer Science
Bishop's University

Sommaire

Cette thèse propose des solutions à quatre nouveaux problèmes issus d'un cadre général de composition horizontale de comportements modélisés à l'aide de systèmes à transition. Ce dernier permet la réalisation d'un nouveau comportement à partir d'un ensemble de comportements prédéfinis, à travers la synthèse d'un contrôleur qui délègue chacune de ses actions à un comportement prédéfini pour son exécution. Dans cette thèse, les comportements sont associés à des composants logiciels, comme des services Web, à des composants matériels, comme des objets connectés, ou à des agents. De plus, une composition est constituée d'un contrôleur et des comportements avec lesquels il interagit pour réaliser un comportement désiré, par exemple celui d'un nouvel agent.

Le premier problème considère que les comportements sont soumis à des contraintes temps réel. La synthèse de contrôleur s'effectue en utilisant les mêmes algorithmes que ceux du cadre général. Toutefois, deux étapes additionnelles sont nécessaires : l'une pour modéliser les interactions entre les comportements et le contrôleur dans une boucle de rétroaction ; l'autre pour vérifier si la boucle de rétroaction est sans interblocage dans toutes ses exécutions considérant l'ensemble des contraintes temps réel.

Le deuxième problème concerne l'assemblage de compositions. Contrairement au cadre général qui utilise des systèmes à transition comme formalisme de modélisation dans un contexte de contrôle purement monolithique, l'approche retenue suggère, d'une part, d'utiliser un calcul de processus comme formalisme pour représenter tous les éléments de la boucle de rétroaction et, d'autre part, d'effectuer un contrôle modulaire c'est-à-dire de combiner des contrôleurs à l'aide d'opérateurs du calcul de processus pour obtenir un contrôle global.

Le troisième problème est une extension du problème de la synthèse de contrôleur lorsque les actions des comportements possèdent des attributs qualitatifs ou quantitatifs et que les actions du comportement désiré sont exprimées sous la forme de préférences. La composition horizontale de comportements basée sur des préférences permet de réaliser un nouveau comportement qui ne pourrait l'être autrement.

Enfin, le dernier problème est celui de la formation d'une équipe d'agents la plus

robuste possible et à moindre coût. Il est formulé comme une problème de programmation linéaire multi-objective en nombre entier. Premièrement, il s'agit de trouver un ensemble de compositions, chacune réalisant le même comportement désiré tout en satisfaisant au mieux ses préférences. Deuxièmement, l'ensemble des agents impliqués dans les compositions forment une équipe qui survit aux pannes d'un ou plusieurs agents.

Cette thèse apporte une solution originale à chacun de ces problème tout en l'illustrant à l'aide d'exemples. L'utilisation des outils SMV/TLV, Uppaal et PuLP permet de vérifier, de synthétiser ou de calculer des éléments des exemples proposés.

Mots-clés: composition de comportements, synthèse de contrôleur, vérification, contrôle modulaire, modèle de préférences, formation d'équipe.

Abstract

This thesis proposes solutions to four new problems stemming from a general framework of horizontal behavior composition, in which transition systems are used to model behaviors. The framework allows the realization of a new behavior from a set of available behaviors, through the synthesis of a controller, which delegates each action of the new behavior to an available behavior for execution. In this thesis, the behaviors are associated with software components—such as web services—, hardware components—such as connected objects—, or even agents. Besides, a composition consists of a controller and the behaviors interacting with the controller for realizing a target behavior, for example the one of a new agent.

The first problem considers that the behaviors are subject to real-time constraints. The controller synthesis is done using the same algorithms as those of the general framework. Two additional steps are, however, required: one for modeling the interactions between the controller and behaviors in a closed-loop control system and another one for checking whether the closed-loop control system is deadlock free in all of its execution according to the set of real-time constraints.

The second problem concerns the assembly of compositions. In contrast to the general framework that uses transition systems as modeling formalism in a purely monolithic control context, the proposed approach, on one hand, uses a process calculus as a formalism to represent all the elements of the closed-loop control system, and, on the other hand, performs a modular control to combine controllers using process calculus operators in order to obtain a global control.

The third problem is an extension of the controller synthesis problem when the operations of the behaviors have qualitative or quantitative attributes and the operations of the target behavior are expressed in the form of preferences. The horizontal preference-based behavior composition makes it possible to realize a new behavior that could not be realized without considering preferences.

Finally, the last problem is the formation of a most robust team of agents at a lower cost. It is formulated as a multi-objective linear integer programming problem. First, it focuses on finding a set of compositions such that each of them carries out the same target behavior while satisfying its preferences at best. Second, all the agents

Abstract

involved in the compositions form a team that remains effective even if one or more agents fail.

This thesis provides an original solution for each of these problems while illustrating it with some examples. The use of SMV/TLV, Uppaal and PuLP tools makes it possible to check, synthesize or calculate the elements of the proposed examples.

Keywords: behavior composition, controller synthesis, verification, modular control, preference model, team formation.

Remerciements

Firstly, I would like to express my sincere gratitude to my supervisor Professor Richard St-Denis for the continuous support of my Ph.D. study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better supervisor or mentor for my Ph.D. study.

Besides my supervisor, I would like to thank the rest of my thesis committee members for their insightful comments and encouragement.

Contents

Contents

List of Figures

List of Figures

List of Tables

Introduction

A software component is a software element [1] that conforms to a component model defining a set of standards for component composition, implementation, naming, interoperability, and deployment. It can be composed individually and deployed without any modification based on a composition standard [23]. Some software components can be qualified as dynamic when considering a dynamic component model in which a means for runtime adaption of a software behavior in response to the environment changes is provided [26]. The recent advances in the domain of component-based software engineering have led to the creation of various kinds of dynamic software components, such as web or cloud services, virtual or abstract IoT devices, and agents.

In the current state-of-the-art of Internet-based technologies, a wide spectrum of dynamic software components is offered as online services or web applications. In many cases, new composite services more usable by such applications need to be created from predefined components. The well-known fundamental problem behind such construction is service composition. However, the way composition is done is still vague in most service-oriented middleware, which give facilities to construct composite services. Furthermore, the composition methods of such middleware still lack formal methods to automatically compose services.

For instance, from a sample of service-oriented IoT middleware reported in three survey papers [5, 43, 53] and recent papers on the subject, more than 50% of them do not suggest any explicit strategy for service composition. About 30% of them advocate the use of a business-process programming language—such as BPEL, iPOJO, Jolie, or PBDL—to express the logic behind a composite service. These languages provide control constructs for structuring workflows. As for the rest of them, an ontology language seems to be the favorite tool to support composition, since semantics is attached to services in that case.

The lack of formal methods for constructing a composite service is not only limited to service-oriented IoT middleware. Research on several solutions proposed for service composition in web and cloud communities [36] shows that most of them are based on

1. A software element includes sequences of abstract program statements, describing computations that must be executed by a machine [23].

informal models and use ad hoc strategies for composing services (e.g., [41, 57, 62]). Moreover, preference is given to the industrial standards WS-BPEL, WSDL, WS-CDL, and OWL-S, which appear as a common denominator for service description and composition on the web [59]. There have been some attempts to apply formal verification methods for service composition (e.g., [33, 63]). Nevertheless, little work has been done on formal synthesis approaches for service composition, despite some recent research in that regard, particularly with applications for web services (e.g. [16, 21, 31]) or services in general (e.g, [14]). Hence, the formal synthesis methods that automatically generate controllers impact on the composite services constructed with service-oriented middleware, because they require that the control be explicit.

One promising synthesis-based model that acts as a formal tool for service composition is a behavior composition framework [32]. It advocates the use of state transition systems as formalism for modeling service behavior. The behavior composition framework has the potential to be used for constructing composite services in recent service-oriented middleware [5]. This sound, complete method provides a technique for synthesizing a controller, which delegates the operations dictated by user requirements—called target service (behavior) and involved in a composite service—to the proper available services (behaviors) being able to offer them in order to realize the target. The original framework is, however, deficient in that it does not take into consideration many characteristics of the services during the calculation of controllers, which act as facilitators in service composition. For instance, from the perspective of the evolution line of services with respect to the current state-of-the-art of IoT and tactile Internet, services attached to IoT and haptic devices must be executed and delivered while satisfying real-time constraints that were not taken into account in the behavior composition framework. Furthermore, a metric to evaluate the degree of match between service operations with semantic similarity, compatibility, and numerical-conformity reasoning functions, or a metric to evaluate the match between the attribute values of operations and preferences of requirements are two examples of aspects that cannot be ignored. As another shortcoming in the framework is modular control. It constitutes an interesting solution when considering multiple target services running in parallel, with or without synchronization between their operations, or a combination of target services with their corresponding controllers

to avoid calculation of a single larger controller.

The behavior composition framework is not only prescribed for the systems or middleware in which the dynamic software components are online services. It can also provide a solution for the automatic composition of other dynamic software components such as agents. Recently, the transposition of a team formation problem into the domain of multiagent systems has led to forming a team of autonomous agents whose mission is to achieve a given goal [47]. When the behaviors of such agents are explicit, an automated team formation problem can be provided by using behavior composition in which a controller is synthesized to delegate the desired tasks of a goal to the available agents that can execute a subset of the tasks. The behavior composition framework is, however, deficient in that as it does not take into consideration some important characteristics of the team formation problem. The calculation of ranks for compositions and the determination of the degree of robustness of a team with respect to the compositions in the first rank or having a rank above a given threshold are two examples of such characteristics that cannot be relinquished. [2]

Objectives

Several new research contributions are essential to achieve or outperform the current state-of-the-art results on the composition of dynamic software components when considering formal methods that emerge from the behavior composition framework, which include synthesis algorithms that automatically compute controllers with the aim of realizing requirements in their entirety by delegating operations. Because the behavior composition framework is the main intended formal tool adopted in this thesis, the following objectives are formulated in terms of questions with respect to this framework and its integration into both service-oriented IoT middleware and the team formation problem.

How can composite services be constructed in service-oriented IoT middleware with the aid of behavior composition? The integration of behavior composition in

2. The degree of robustness is the number of agents that can be removed from a team such that the team remains effective, namely satisfies a given goal.

service-oriented IoT middleware requires a reexamination of the composition layer of a middleware architecture to automatically construct composite services. To this end, some new components should be introduced in the layer to synthesize and verify such composite services. The revisited architecture, in turn, underscores the necessity of several extensions in the original behavior composition framework to support some characteristics of service operations. The real-time constraints imposed by some operations in IoT applications is an example of such characteristics.

How can the real-time constraints imposed by service operations be taken into consideration in the behavior composition framework? Facilities to support temporal constraints in the service-oriented IoT middleware are quite significant to guarantee efficiency in delivery of services, particularly when they are associated with haptic devices, which recreate the sense of touch. In such technologies, the parameter of time for communication between a master agent and a slave one is so important. The notion of time is absent in the finite-state transition systems defined in the original behavior composition framework. First, this question subsumes the simulation of timed models, particularly the interactions among the main elements of the framework, namely a target behavior, a controller, and available behaviors. Second, it subsumes the verification of formulas expressed in a subset of temporal logics that are available in model-checking tools.

How can a modular control be provided for the behavior composition framework? This question is related to a particular modular composition problem in this context. Originally, the behavior composition problem was formulated for a single target. Thus, there was only one controller to synthesize. An extension has been proposed for multiple targets [55], but the underlying solution is inadequate when it is transposed in the web communities, where many requests must be served simultaneously and various ways to synchronize or not the targets must be considered. In fact, the proposed solution is limited because it ignores the deadlock issue, and synchronization consists of adding a unique behavior (causing a bottleneck) that coordinates targets through input–output messages explicitly added in the latter. A richer solution consists in paying attention to several forms of structures or combinations among the targets defined by the user requirements, such as concatenation, choice, iteration, and so forth. This, however, requires investigating relationships between a composite controller and

the controller derived from the corresponding composite target, because they are in general different, especially when taking into account the degree of match or weight metrics in the synthesis algorithms.

How can the degree of match between service operations be defined in the behavior composition? In the original behavior composition framework, the operations requested by the target behavior must be the same (i.e., have the same names) as those executed by the available behaviors. The operations must be synchronized in the controller synthesis procedure. In fact, interchanging operations (i.e., operations having different names, but the same functionality) are not allowed. More specifically, in semantic-based systems, such as the recent search engines designed for cloud computing [57], interchanging service operations, viewed as matchable (or similar), is fully supported. This question is thus raised when integrating behavior composition as a formal tool into such systems.

How can the preferences of user requirements be supported in the behavior composition framework? In service-oriented computing systems, such as the cloud, most service providers determine a set of attributes (e.g., cost and type) for service operations. Besides, the users of such services may express some qualitative and quantitative preferences in their requirements. A question that can be raised when behavior composition is integrated into such systems is how such preferences can be matched with the attribute values of operations through the controller synthesis algorithms proposed in the framework.

How can the behavior composition framework be integrated into a team formation problem? A generalization in a typical robust team formation problem consists in defining behaviors for agents, assigning attributes to the tasks of agents, and attaching preferences to the tasks of a goal. Such a generalization motivates the integration of behavior composition into the team formation problem to automatically make a composition involving a controller and a team of agents performing the tasks of the goal. To this end, an extension in the behavior composition framework is required to support preferences of the goal. The details about such an extension was mentioned in the question related to the previous objective. In integrating the framework into the team formation problem, some utility values, such as cost, can be assigned to the compositions. Hence, a main question that can be raised is how a robust team with

better values for compositions can be found.

Methodology

To reach acceptable solutions as answers to the questions stated in the previous section, the methodology has been organized in phases, one per question. For each phase, when it is appropriate, several implementations and simulations complement the theoretical developments. The model-checking tools SMV and TLV are used to implement synthesis algorithms. Specifically, the modules for defining the elements of the behavior composition framework are specified in SMV and the controller synthesis procedures are defined with TLV. Modules and procedures are generally adapted with respect to the concepts introduced in each phase. In the case of defining a timed model for the behavior composition framework, several properties such as reachability and deadlock can be verified on controllers through Uppaal. It is a real-time model-checking tool supporting a subset of TCTL and WCTL temporal logics [40].

Phase 1: A revision in the service-based IoT middleware architecture—The evolution of service representations shows that they can be classified into three generations. Hence, three abstract models can be introduced to indicate the evolution of service representations. The composition of services with such abstract models requires a re-examination of the service-oriented IoT middleware. To this end, the following steps are considered.

- Examine a recent service-oriented IoT middleware architecture designed in [5].
- Introduce a description about the abstract service models each of which refers to a generation of service representation.
- Integrate the abstract models into the aforementioned middleware and reexamine the service composition layer of the architecture to construct composite services with the aid of the synthesis method included in the behavior composition framework.
- Introduce some new components in the architecture to enable verification of composite services and support the closed-loop control of haptic communications in the middleware.

Phase 2: A timed-model of behavior composition (simulation and verification)—After the synthesis of a controller (orchestrator) in the behavior composition framework, some interactions are performed among a target behavior, the orchestrator, and available behaviors. If both the target and available behaviors are subject to time constraints for operation execution, the simulation and verification of the interactions require the following steps.

- Define both target behavior and available behaviors in the form of timed automata.
- Synthesize orchestrators by disregarding the time constraints imposed by the target behavior and available behaviors.
- Introduce a closed-loop control system for the interactions among the target, a synthesized orchestrator, and available behaviors.
- Simulate the closed-loop control system in UPPAAL and then verify the interactions to detect feasible deadlocks or check temporal properties.

Phase 3: Modular composition of multiple targets and controllers—A process-theoretic approach is the foundation to study modular behavior composition, more specifically, to formulate various modular composition problems and suggest solutions for all of them, while considering the degree of match and weight metrics. In this approach, the main elements of the behavior composition framework (i.e., target behavior, available behavior, system, and controller) are formalized as process terms of a process calculus. Therefore, the following steps are considered.

- Reformulate the main elements of the behavior composition framework as process terms.
- Introduce a process calculus with different types of synchronization operators depending on the execution schema adopted to synchronize operations requested by multiple targets and executed by one or more multi-threaded or not behaviors, while considering global ordering constraints on operations.
- Propose a new algorithm for the automatic construction of controllers.
- Verify that the set of process operators that can be placed among the targets is compatible with the corresponding controllers.

– Define ternary process-synchronization operators between target, system, and controller processes.

– Establish theoretical relationships between the controller of composite targets (e.g., $t_1 \odot t_2$) and the composite controller of the corresponding targets (e.g., $o_1 \odot o_2$), where "$\odot$" is any acceptable process operator between targets, since their weight may differ.

Phase 4: Degree of match between operations—The definition of operations in the behavior composition framework can be enriched by attaching semantic attributes to the operations through ontologies. The following steps outline this phase, which provides a semantic-based framework for behavior composition.

– From ontologies, consider similarity graphs in which a node represents an operation and an edge indicates a relationship between two operations (e.g., is-a).

– Based on a semantic similarity function defined from a similarity graph, like the one introduced in [3] or [34], compute coefficient values of a similarity matrix $n \times n$, where n is the number of predefined operations.

– Find or define other resource reasonings between operations to enhance the behavior composition framework such that it better matches the reality of semantic-based systems. For instance, in the *Cloudle* architecture designed for the agent-based search engines in the cloud, three types of resource reasoning are implemented with respect to an ontology graph: similarity, compatibility, and numerical reasoning [57].

– Adapt the definition of the largest ND-simulation relation and the definition of the controller generator as originally defined in the behavior composition framework to take into consideration equivalence conditions between operations. An example of such a condition is the similarity condition $sim(a_t, a_s) \geq \mathit{Threshold}$, where a_t and a_s are the operations requested by a target behavior and executed by an available behavior, respectively.

– Revisit the synthesis algorithms that calculate controllers (or compositions): one based on the notion of largest ND-simulation relation and one based on the concept of safety game.

Phase 5: Preferences and attributes—Each operation of an available behavior in the behavior composition framework can be assigned with a set of attributes. Furthermore, some preferences can be expressed through a target behavior. The following steps outline this phase, which provides a preference-based framework for behavior composition.

- Examine the work reported in [38], which introduces a preference model for supporting complex database queries to match user preferences.
- Revisit the representation of available behaviors and target behavior in the original behavior composition framework to take into consideration attributes and preferences.
- Adapt the definition of the largest ND-simulation relation and the definition of the controller generator as originally defined in the framework to take into account the match between attribute values and preferences.

This phase can be extended to combine the notion of resource reasoning and preference models in the original behavior composition framework. Such a combination gives rise to a hybrid framework, namely, a semantic-supported and preference-based framework that not only matches similar operations and similar attributes, but also similar attribute values and preferences. In this end, the following extension is required to achieve such a hybrid framework.

- Define new quantitative preference models for similarity, compatibility, and numerical reasoning.
- Revisit the largest ND-simulation relation and the controller generator definition to match the attribute values and preferences with respect to the new preference models defined for resource reasoning.

Phase 6: Behavior composition and team formation problem—According to an attributed multiagent system and a goal with preferences, a set of compositions is found that realize the goal. The agents involved in these compositions form the more robust team at lower cost. This is a reformulation of a team formation problem through preference-based behavior composition, whose realization consists of the following steps.

– Examine the work reported in [47] that introduces a formal framework for a robust team formation problem and provide some solutions for optimizing the robustness of a team, while maintaining its cost below than a given value.

– Integrate the preference-based behavior composition into the robust team formation problem, which involves the representation of each agent as an available behavior and a goal as a target behavior. In this step, a controller synthesized for the realization of the goal along with the agents executing a subset of tasks in the goal constitute a composition.

– Rank the compositions based on their fitness with respect to the qualitative and quantitative preferences. In this step, an efficient nondominated sort algorithm, proposed in [25] is used for ranking of compositions.

– Assign some global parameters to the compositions and use a mathematical programming technique to solve an optimization problem according to the parameters.

Results

Based on the phases defined in the methodology section, the following original results have been achieved.

Phase 1: A revision in the service-based IoT middleware architecture—The first result of this phase is the abstract service models that describe the evolution of service representation, ranging from a traditional web service to a composite service constructed for haptic devices. The next result is a schema of an architecture designed for service-oriented IoT middleware, which supports the composition of services described by the abstract models.

Phase 2: A timed-model of behavior composition (simulation and verification)—The result of this phase constitutes a threefold contribution. First, it defines a timed transition structure for each behavior in the behavior composition framework. Second, it provides a closed-loop control system for the interactions, which are performed among the main elements of the framework. Third, the closed-loop control system is simulated in Uppaal and verified through several kinds of properties that involve time constraints.

Phase 3: Modular composition of multiple targets and controllers—Regarding multiple targets as well as the way of combining them and combining their controllers with respect to several operators (e.g., concatenation, choice, iteration, interleaving), a modular behavior composition method based on a process calculus with well-defined semantic rules is introduced as the main result of this phase. This phase states theorems and establishes proofs about the relationships between the controller of a composite target and the corresponding composite controller. When such relationships do not stand, counterexamples are provided.

Phase 4: Degree of match between operations—The results of this phase can be extracted from [11]. In this paper, an ontology graph and a semantic similarity function are used to consider similar operations in the behavior composition framework. Behavior composition has also been adapted to the other types of resource reasoning: compatibility and numerical [8]. Similar or matchable services have been implemented in SMV/TLV.

Phase 5: Preferences and attributes—The result of this phase can be extracted from [12]. In a part of this paper, the behavior composition framework is revisited to support operation attributes and preferences. A new version of the largest ND-simulation relation is proposed that not only matches operations but also the attribute values and qualitative preferences. Then, the original SMV module skeletons, which only support functional properties (operations), were modified to include nonfunctional properties (preferences and attributes) by exploiting set operators. Furthermore, the other result of this phase is the hybrid framework of behavior composition, supporting both resource reasoning and preferences.

Phase 6: Behavior composition and team formation problem—The result of this phase can be also extracted from [12]. This paper generalizes a robust team formation problem by assigning behaviors to agents whose tasks are equipped with multiple attributes. Their values are compared with preferences attached to the tasks of a goal. In this paper, a constraint-optimization problem is formulated such that its objective function is defined by considering cost and degree of robustness. Furthermore, some experiments with a synthetic multiagent system are provided to indicate the impact of composite preferences on the distribution of compositions with respect to their rank and degree of robustness.

Organization

The rest of this thesis is structured as follows. Chapter 1 revisits a service-based IoT middleware architecture to automatically compose IoT devices abstracted by a dynamic behavioral description. Chapter 2 provides the primary concepts about the original behavior composition framework used as a starting point for the composition of dynamic software components, while laying stress on its drawbacks. Chapter 3 defines a time model for the behavior composition framework and describes the implementation of some of its elements in Uppaal. Chapter 4 provides horizontal modular control for the behavior composition framework through a process-theoretic approach. Chapter 5 introduces a semantic-supported and preference-based framework for behavior composition, which involves the notions of resource reasoning and preference models. This chapter consists of the solutions for Phases 4 and 5 mentioned in the methodology section. Finally, Chapter 6 is about integrating the preference-based behavior composition framework into a team formation problem.

In the next chapters, some elements of the original behavior composition framework may appear with different terminologies. Appendix A provides a facility for readers to access these terminologies.

Chapter 1

Abstract Services in IoT Middleware

Software engineering practices have constantly been adjusted to comply with the evolution of modern software applications. This is the case for IoT applications that are arriving at a turning point at which the level of complexity further motivates the importance of middleware. Given available IoT middleware, the service-oriented ones suggest that IoT devices, with real-time functionalities accessible through an API, take the form of atomic services. Given this context, service-oriented middleware should ease the development, deployment, configuration, and monitoring of adaptable and reliable real-time IoT applications. To be valuable, they must be merged with service-based architectures. Indeed, service-based architectures have the potential to combine IoT and real-time communication (RTC) in such a way that a device can be linked to a real-time application and firmly provide responses within given deadlines. The current WebRTC project, which has a close relationship with Object RTC, is an attempt in this direction [39]. Besides, a lot of research effort has been invested in the study of haptic communication aspects related to the physical and network layers of service-based architectures specific to IoT applications [1]. Given the wide spectrum of services created from IoT devices, new composite services more usable by IoT applications need to be created from them. The well-known fundamental problem behind such construction is the composition of services.

During the fast growth in Internet technologies, IoT users are witnesses of a tangible evolution in the service representations. An exploration about such representations figures out they can be classified in three generations. In this end, three abstract models of services can be introduced to illustrate the evolution in their representations. However, given such abstract models, the composition of services requires a reexamination in the service-oriented IoT middleware. Introducing of such abstractions for services and providing a solution for composition of them in the middleware consist in the following phases:

– The clues about the abstract service models, which highlight communication and control, are provided.

– The service composition layer of a service-oriented IoT middleware architecture is revisited to support the abstraction of services and automatically compose them. By doing so, IoT devices are represented by state transition systems to lay the foundations for synthesis and verification.

1.1 Internet of Things

The internet of things (IoT) is the network of physical devices, which are embedded with sensors, actuators, software, and electronics. Such devices are monitored or controlled from a remote location through Internet [5]. There are several definitions about IoT, for instance, one of them describes IoT as a global infrastructure for the information society, which enables advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies [61].

With the fast growth of IoT devices, it is forecast by 2020 that about 26 billion devices, ranging from smart pens to heart monitoring implants, will be connected to Internet [37]. Hence, IoT users will be provided with a lot of applications and services in the near future. Such applications have the following characteristics [53].

Diverse applications—IoT offers its services to the numerous applications in various domains. The domains are categorized into smart environments, healthcare, transportation, industrial, and personal domains. Figure 1.1 shows some main application domains for IoT.

Real-time—IoT applications are classified as real-time and non-real-time. The real-time ones refer to the applications, which provide a response within a given deadline. For instance, both IoT healthcare and IoT transportation applications can be considered as real-time.

Everything-as-a-service (*XaaS*)—With the growth of connected devices, the collection of services is likely to rise. Since such services are accessible online, they are available for use and reuse. Totally, an XaaS model is very scalable, efficient, and easy to use.

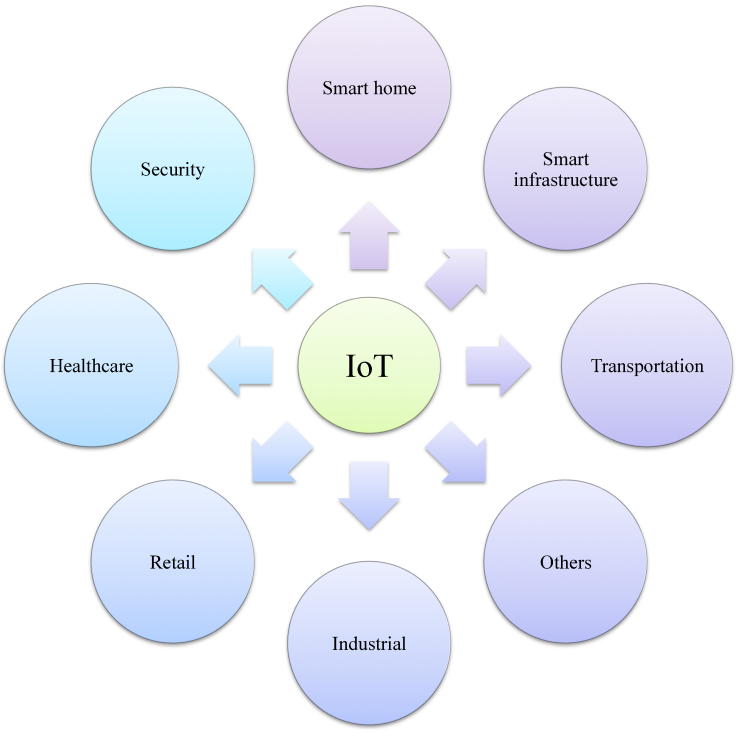


Figure 1.1: Applications of IoT

Increased security attack-surface—As many as there are a lot of IoT applications in various domains, there are also concerns about the security of such applications. IoT requires global accessibility and connectivity, that is, each user is able to access it anytime and anyway. This drastically raises the attack surfaces for the IoT applications and networks.

Privacy leakage—Some IoT applications are made to collect the information about people activities, which are considered as private and should not be exposure. By doing so, any IoT application, which is not compliant with privacy requirements can be forbidden by law.

As depicted in Fig. 1.1, the potential application of IoT is vast, and such applications have been permeated into all areas of our daily life, society, and industries. For instance, in the domain of healthcare, there are some smart health applications listed as follows [61].

- *Hygienic hand control*: It includes an RFID-based monitoring system of wrist bands in combination with Bluetooth tags on a patient doorway, which control the hand hygiene in a hospital. In the warning situation, where the doorway is not hygienic, a vibration notification is sent out to inform the patient about hand washing.
- *Fall detection*: It is utilized for disable or elderly individuals, who are living with a certain autonomy.
- *Physical activity monitoring for aging people*: It is implemented by a body sensor network, which measures vital signs and motion, and a mobile unit, which collects and records data.
- *Chronic disease management*: It is a patient-monitoring system for remote monitoring of patients with chronic diseases such as diabetes.
- *Dental health*: It is a bluetooth connected to a toothbrush along with a smartphone app that analyzes the brushing uses and can give some necessary information to a dentist, remotely.
- *Patients surveillance*: It monitors the conditions of a patient in a hospital or monitors an elderly people in a home.
- *Medical fridges*: It controls the conditions of the freezers, which store medicines and vaccines.

1.2 IoT Middleware

A middleware is a software layer or a set of sub-layers, which are interjected between the application levels and the technological levels. The feature of a middleware is hiding the details about the technologies exploited in its layers [5].

IoT middleware are categorized in seven groups: event-based, service-oriented, VM-based, agent-based, tuple-spaces, database-oriented, and application-specific [53]. *Event-based middleware*—In this kind of middleware, applications, components, and other participants have interactions via events. The model of publish/subscribe is used and permits the subscribers to access publishers data through a shared database. The subscribers are registered for specific events.

Service-oriented middleware—In this kind of middleware, user applications and software are made in the form of services. The architectures adopt the service-oriented architecture (SOA) and permits developers to deploy numerous of IoT devices as services.
VM-based middleware—In this kind of middleware, a safe execution environment for applications and also programming supports are provided through virtualization of the infrastructure and network. By doing so, the user applications are divided into small modules distributed in the network. Every node in the network is equipped with a virtual machine, interpreting the modules.
Agent-based middleware—In this kind of middleware, applications are partitioned into modular programs to simplify distribution of them with the aid of mobile agents. An agent can communicate with other software agents to obtain data and update just the part of an application, assigned to it.
Tuple-spaces middleware—In this kind of middleware, a local tuple space structure is held by each member of the infrastructure. A data repository, which can be concurrently accessed, is called a tuple space. A federated tuple space is formed through all the tuple spaces. Applications interact with writing tuples in a federated tuple space and with reading tuples via identifying the pattern of the data, which are inserted in the federated tuple space.
Database-oriented middleware—In this kind of middleware, a sensor network is seen as a virtual relational database system. A user application is able to query the database by using a query language like SQL. Easy-to-use interfaces support user queries to the sensor network for extracting data.
Application-specific middleware—In this kind of middleware, the focus is on QoS and resource management supports for applications. The latter permit applications to identify their QoS requirements and provide a runtime adjustment for the network configurations based on the requirements.

1.2.1 Service-Based IoT Middleware Architecture (SIMA)

The service-oriented IoT middleware not only support the abstraction of devices but also support service management through service compositions. In such middlewares, the functionalities of devices are provided as services and are stored in service

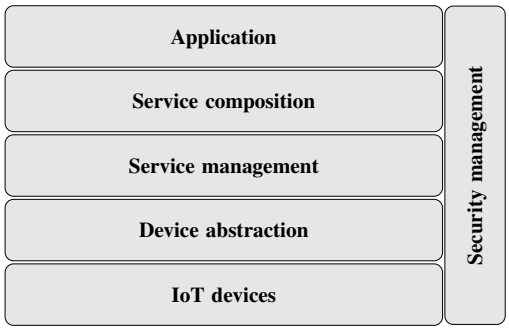


Figure 1.2: Service-based IoT middleware architecture

repositories. These services are atomic services and they are the links between the physical world and the traditional service-oriented systems [42].

Figure 1.2 illustrates a general service-oriented IoT middleware architecture [5]. The architecture consists in six layers summarized as follows.

IoT devices—This layer encompasses a variety of connected physical devices, ranging from personal ones such as tablets, smart phones, and digital cameras to industrial ones such as motors, sensors, machines, and robots.

Device abstraction—This layer, called also device management, provides the capability of access to different devices through a common standard language.

Service management—This layer manages the main functionalities of devices in the form of services. A repository is built in the layer to register the catalogue of services associated to each device.

Service composition—This layer supplies the capability of composing services offered by connected devices in order to make specific applications. In the layer, the notion of devices is disappeared and the only visible entities are services.

Application—This layer is at the top of the architecture, which exports the functionalities of the system to the IoT users.

Security management—This layer contains the functions applied for the management of the privacy, trust, and security of all exchanged data throughout the middleware.

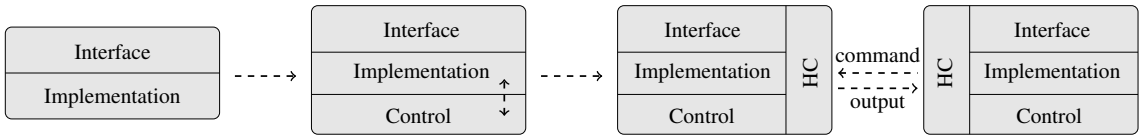


Figure 1.3: Evolution of abstract representations of services

1.3 Abstract Service Models

An abstract service model is defined as a representation of a few basic constituent elements with their relationships, which is intended to understand how a service operates. Instances of abstract service models can be seen as small boxes that can be combined horizontally and vertically in a hierarchical manner [22]. Based on the current trends and rapid acceleration in Internet technologies, a tangible evolution over three generations of such models can be observed. Figure 1.3 shows three general models. A review of the literature reveals that most Internet-based services can be associated with one of them.

Starting with the first generation, along with the basic notions of interface and implementation, subsequent generations successively add the notion of control, then the notion of haptic communication (HC), to the preceding one. Contrary to a first-generation service, a second-generation service is equipped with a controller that explicitly acts on service operations that are invoked to realize a correct implementation with respect to some control requirements. The control logic is clearly separated from other implementation details instead of being scattered in the implementation as is usually the case in most current practices of programming [21]. A third-generation service enables, among other things, the capture of data on touch communication through haptic technologies developed via the tactile Internet [1].

In a typical instance of the first-generation model, the implementation component is developed by using a programming language such as WS-BPEL. The list of available operations is exhibited through the interface component specified in WSDL and published in a UDDI directory. OWL-S can also be used when more semantics is attached to services [45]. Most of the middleware proposed around this model has some limitations, mainly because the public service description does not include a

service usage protocol, which indicates the order in which its operations must be invoked [46]. Essentially, a WSDL file is like a functional specification of the operations offered by the service. Without a well-defined service usage protocol, it is not clear how to restrict or manage the invocation of operations to ensure that only admissible execution traces will result from the code provided in the implementation part.

The second-generation model specifically addresses the aforementioned point. The addition of an explicit controller, however, changes the way a service is specified. Its description includes both a functional specification and a usage protocol. The latter exhibits behaviors that are semantically defined in terms of admissible execution traces. It allows for automatic synthesis of controllers, which control the flow of operations. It is important to mention that nondeterminism inherently arises in a service usage protocol because it is an abstraction of an implementation. For instance, invisible operations outside the implementation part are hidden. Thus, they do not appear in the usage protocol. There are two main kinds of control: delegation and supervision. Accordingly, a controller is called an orchestrator or a supervisor.

- An orchestrator delegates each requested operation to a service available to execute it, as introduced in the behavior composition framework [32].
- A supervisor enables or disables controllable operations through control actions, as defined in the supervisory control theory [51].

The third-generation model comes with the recent emergence of tactile Internet, with haptic communication as the primary application. In fact, tactile Internet is the medium for transmitting sensing data as outputs and commands as inputs. The exchange of information between a composite service (the rightmost box in Fig. 1.3) and an IoT device (the second box from the right in Fig. 1.3), through its sensors and actuators, is established inside a distributed closed control loop in which the controller of the composite service sends commands (feedback control) based on the device's current output (feedback signal). To achieve very low end-to-end latencies, the communication part must have capabilities, as never reached before. Without such capabilities, the closed-loop control system may be unstable under time-varying delays.

Although IoT has already had undeniable impacts on the business community [60], the services largely obey the principles conveyed by the first generation. A good ex-

ample is a list of individual services specific to an IoT device integrated into a unit service node (an abstract service model) [48]. The notions of closed-loop control and haptic communication are absent in this model. A composite service is implemented from the composition of unit service nodes with the aid of a directed graph formalism without guarantee of correctness with respect to behavioral and real-time requirements.

Eventually, the evolution of abstract service models described in this section should converge to more advanced ones, reflecting the advances in IoT, and thus make services more intelligent.

1.4 Impacts of Abstract Service Models on SIMA

The adoption of an unconventional abstract service model of the second or third generation depicted in Fig. 1.3 has some impacts on service-based IoT middleware architectures. The aim of presenting such an architecture is to highlight the interacting components that are more related to control, especially those providing support for automatic synthesis of controllers.

Figure 1.4 shows some of the main components available in the key layers of the architecture depicted in Fig. 1.2. Notably that, the security management layer is disappeared in Fig. 1.4, since the security aspect is not a concern in the revisited architecture.

The central component is the abstract service pool located in the service composition layer. It is filled following upward and downward flows of interactions from the device management layer and application layer, respectively. Atomic services and composite services are built following the upward and downward flows, respectively. These services should conform to the aforementioned abstract service models. Notably, unlike an atomic service, the method of composition in a composite service is explicit [29].

The details about the main layers of the architecture along with their components are provided as follows.

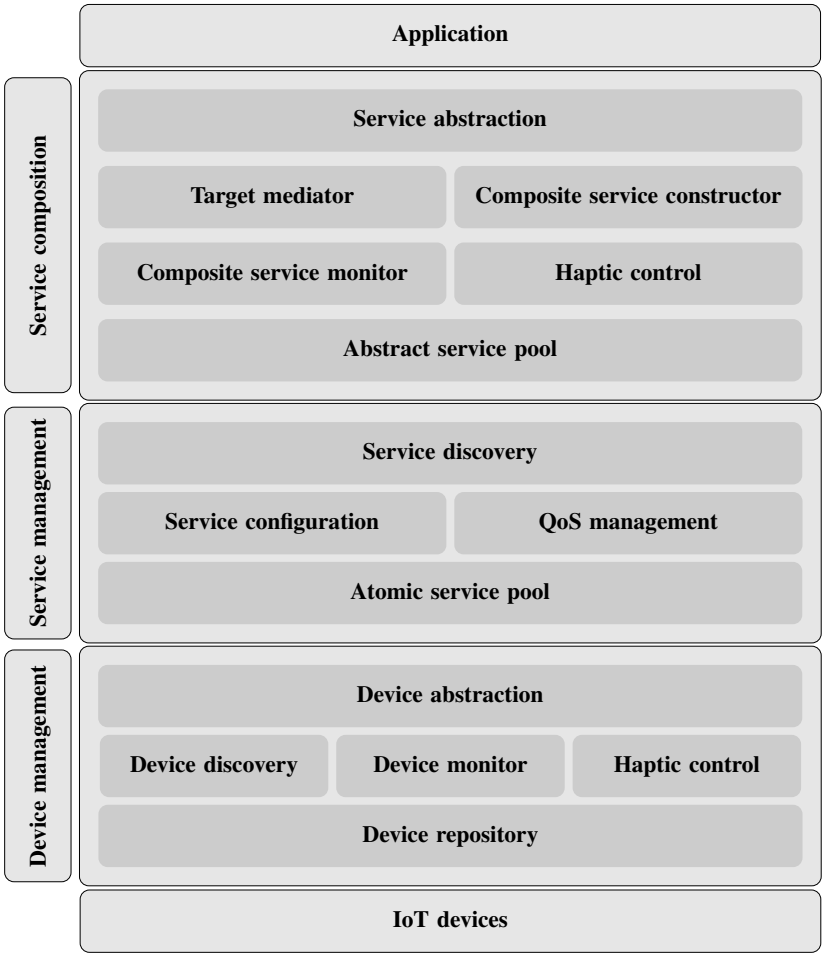


Figure 1.4: A service-based architecture supporting abstract models

1.4.1 Device Management Layer

This layer contains five components: device abstraction, device discovery, device monitor, device repository, and haptic control. Ideally, IoT devices should directly export their services. Some of them are, however, unable to present their functionalities as services, analogous to actuators or low-end devices using a data-centric communication or a message-passing mechanism. In this case, the device abstraction component supplies the capability of wrapping functionalities around a service representation through the provision of both a web service interface and an implementation

part. The former facilitates the access of connected devices by using a common standard language. The latter translates some service operations into compatible device commands and captures sensing data as a result value to be returned by a service query operation. The device discovery component finds the available IoT devices over the network, and the information about the status of such discovered devices (e.g., failure detections) are maintained through the service monitor component. Besides, the details about the types, constraints, and features of existing devices are registered in the device repository. In this layer, the haptic control component provides facilities for establishing haptic communication in order to implement a closed-loop control system [1].

1.4.2 Service Management Layer

The visible assets in this layer are just services, being abstractions of operations handled by IoT devices or services discovered from existing distributed repositories. The main components of the layer are: service discovery, service configuration, QoS management, and an atomic service pool. In some service-oriented middleware architectures, the layer includes also other components such as data management, lock management, and service monitor [58]. The service discovery component can have both passive and active modes. In the passive mode, the service discovery component only listens to the services, which are publishing from the device management layer. In the active mode, it not only discovers the ones produced by connected devices or offered largely by distributed repositories but also retrieves the information about the services released over the world wide web by using some web crawlers. All the discovered services are stored in the atomic service pool provided in the layer. Furthermore, they can be configured and installed via the service configuration component. Finally, the role of QoS management component is keeping all details about the quality aspects of services, namely cost, availability, time constraint, and so forth.

1.4.3 Service Composition Layer

This layer is different from those provided in analogous middleware architectures mainly because the construction of composite services is supported by a formal synthesis/verification method, which requires a dynamic behavioral description of services.

Since many of the components of this layer are novel, they must be explained in more detail. The focus is put on composition by delegation.

Service abstraction—This component facilitates the creation and updating of formal specifications in the form of transition structures peculiar to the synthesis/verification procedure for atomic, composite, and target services (i.e., requirements for a new service). If the description of a service includes a service usage protocol [46], conversion procedures help to produce the corresponding transition structure. The transition structure formalism depends on the type of service to be described: ordinary service (O-service), real-time service (RT-service), and haptic service (H-service). For instance, the timed automaton formalism is appropriate for modeling RT-services and H-services, since it includes clock variables and clock constraints on transitions and clock invariants on states.

Target mediator—This component plays the role of a broker to handle target services. On receiving such a service, it identifies the corresponding composite service, if it exists in the pool.

Composite service constructor—This component is the heart of this layer. It encompasses a formal synthesis/verification procedure. Synthesis allows for automatic construction of an orchestrator, which combines services such that the resulting composite service fulfills the behavioral requirements of a target service. Verification allows for checking if the constructed composite service satisfies the real-time constraints imposed on the target service.

Composite service monitor—This component is responsible for maintaining up-to-date information on costs and response times about services. It also offers recovery facilities for orchestrators in case of service failures.

Haptic control—This component is the counterpart of the similar one introduced in the device management layer to support the implementation of a closed-loop control system.

Abstract service pool—This component stores atomic and composite services with all information associated with them. In particular, each composite service has a target service. The former can be reused by the target mediator. It can also be combined with other atomic and composite services, in a hierarchical way, to realize more complex targets.

1.4.4 Application Layer

This layer can offer two different types of interface. One is considered for the purpose of entering the requirements and facilitating the interactions between the IoT users and the published composite services. The other one, called haptic user interface, is considered for supporting haptic communications in which the user inputs are converted to the tactile ones by the aid of several tactile programming techniques.

1.4.5 A Short Scenario

The following scenario illustrates the downward flow of interactions to acquire a composite service from some user requirements. The gathering of atomic services ready to be stored in the same pool follows the conventional upward flow.

Let us assume that functional requirements for a target service are provided from the application layer. First, the service abstraction component helps to build the transition structure according to the type of service. Second, the target mediator searches for a composite service in the abstract service pool that matches the description of the target service. If there is no match, the composite service must be created with the aid of the composite service constructor. After identifying the available services in the pool that can be useful to realize the target service, the orchestrator is automatically calculated and integrated into a new composite service. Several parts of its implementation can be automatically produced from the description of available services and the transition structure of the target service. Its interface is obtained from an abstraction of the latter. If the requirements include real-time constraints, the composite service constructor also verifies if the composite service satisfies them. Third, the new composite service is stored in the pool and its reference is sent to the target mediator, which forwards it to the application layer.

1.4.6 Realization of an Intended Component

In order to realize all components of the service composition layer depicted in Fig. 1.4, a huge amount of work is required. The focus of this thesis is, among other things, on the composite service constructor component, since it is the heart of the layer. The clues about the realization of the component consists in the following

phases:

- A solution for synthesizing an orchestrator for a composite service is provided with many extensions of the behavior composition framework.
- The interactions between the composite service under control and available atomic services form a closed-loop control system, which is systematically implemented in a model checking tool, namely UPPAAL. The execution of a composite service can then be simulated.
- Several verifications are done on a composite service in UPPAAL with respect to some properties, including safety, liveness, reachability, and deadlock.

1.5 Contributions

In summary, this chapter makes the following contributions.

- It introduces a better description of three abstract service models, each of them corresponding to a specific generation of services considering their recent evolution.
- It integrates the abstract models of services into service-oriented IoT middleware to define a dynamic behavioral description for IoT devices.
- It reexamines the service composition layer of a service-oriented IoT middleware architecture to construct composite services with the aid of a formal synthesis/verification method.
- It introduces new components in a service-oriented IoT middleware architecture to support haptic communications and the implementation of a closed-loop control between a haptic user and a master, using haptic devices.
- It introduces an abstract service pool in the service composition layer of the architecture to provide an upward and downward flows of interactions from the device management layer and application layer in the architecture. Then, both atomic and composite services are built through such flows.

Chapter 2

Preliminaries—a Behavior Composition Framework

A typical behavior composition problem consists in the synthesis of a controller in order to realize a desired target behavior by coordinating a set of available behaviors. This problem was studied in [32] through which an original and substantial framework, called *automatic behavior composition*, was proposed. The aim of this chapter is to present this framework. In this framework, each behavior is an abstract model of an agent, device, or software component operating on an environment, which is a shared space where actions are defined. The behavior composition problem has been also formulated in various contexts: web services [14], verification [44], robotics [18], and even multi-agent systems [52].

Since the main elements of the behavior composition framework are defined by transition systems, Table 2.1 provides a facility for readers to access some notations used in this chapter.

Table 2.1: The notations used in the behavior composition framework

Element	Transition system	State	Transition relation
Environment	$\mathcal{E}$	$e \in E$	ρ
Behavior	$\mathcal{B}$	$b \in B$	δ
Available behavior	$\mathcal{B}_i$	$b_i \in B_i$	δ_i
Target behavior	$\mathcal{B}_t$	$t \in B_t$	δ_t
System	$\mathcal{S}$	$s \in S, s = \langle b_1, \ldots, b_n \rangle$	δ
Enacted behavior	$\mathcal{T}_{\mathcal{B}}$	$\boldsymbol{b} \in S_{\mathcal{B}}, \boldsymbol{b} = \langle b, e \rangle$	$\delta_{\mathcal{B}}$
Enacted target behavior	$\mathcal{T}_{\mathcal{B}_t}$	$\boldsymbol{t} \in S_{\mathcal{B}_t}, \boldsymbol{t} = \langle t, e \rangle$	$\delta_{\mathcal{B}_t}$
Enacted system behavior	$\mathcal{T}_{\mathcal{S}}$	$\boldsymbol{s} \in S_{\mathcal{S}}, \boldsymbol{s} = \langle b_1, \ldots, b_n, e \rangle$	$\delta_{\mathcal{S}}$
Controller generator	CG	$\sigma \in \Sigma, \sigma = \langle \boldsymbol{t}, \boldsymbol{s} \rangle$ if environment $\sigma \in \Sigma, \sigma = \langle t, s \rangle$ if no environment	ξ

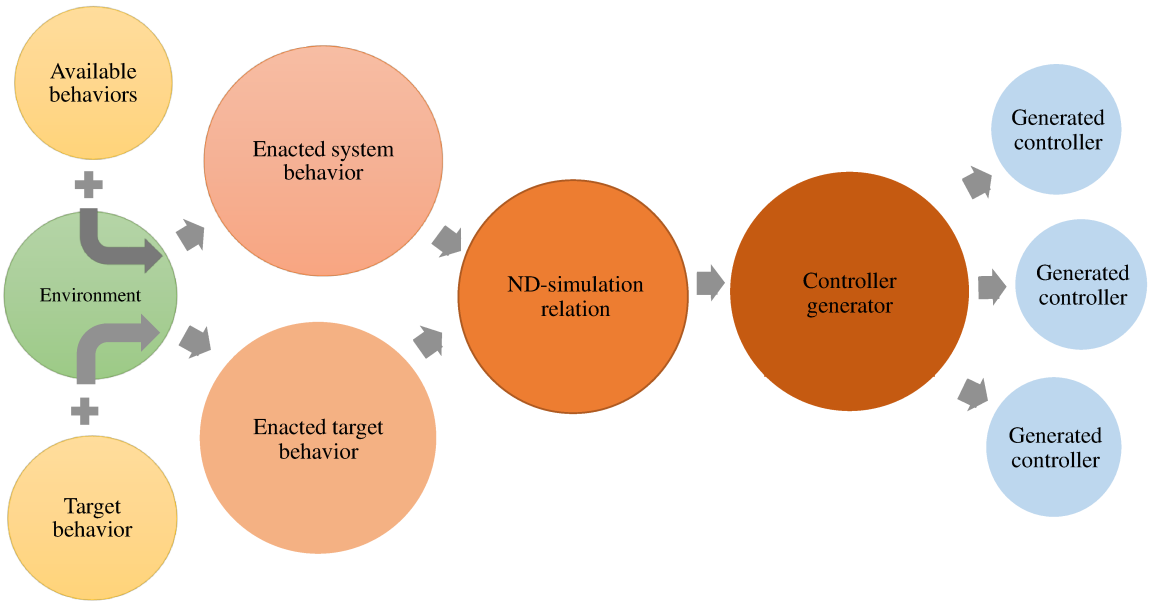


Figure 2.1: The behavior composition framework

2.1 Original Behavior Composition Framework

Figure 2.1 illustrates a general view of the behavior composition framework. The main elements of the framework are environment, available behaviors, target behavior, controller generator, and generated controllers.

Definition 2.1.1. An environment, which is generally nondeterministic, is a tuple $\mathcal{E} = \langle A, E, e_0, \rho \rangle$, where A is a finite set of shared actions, E is the finite set of environment states, $e_0 \in E$ is the environment initial state, and $\rho \subseteq E \times A \times E$ is the environment transition relation.

It should be noted that, a guard over $\mathcal{E}$ is a Boolean function $g : E \mapsto \{\top, \bot\}$. Guards are used as a means to impose conditions on the evolution of behaviors with respect to the current state of $\mathcal{E}$.

Definition 2.1.2. An available behavior, which is generally nondeterministic, is a tuple $\mathcal{B}_i = \langle B_i, \delta_i, b_{i0}, G_i, F_i \rangle$, where B_i is the finite set of behavior states, $\delta_i \subseteq B_i \times G_i \times A \times B_i$ is the behavior transition relation, $b_{i0} \in B_i$ is the behavior initial state,

G_i is a set of guards on an environment $\mathcal{E}$ with a set of shared actions A, and $F_i \subseteq B_i$ is the set of behavior final states.

Similar to Def. 2.1.2, a target behavior $\mathcal{B}_t$ is a tuple $\langle B_t, \delta_t, b_{t0}, G_t, F_t \rangle$, whereas, on the contrary to an available behavior, $\mathcal{B}_t$ is deterministic. A target behavior indicates the fully controllable desired behavior to be reached.

Given the available behaviors and an environment, a system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ is defined as the interleaving (composition) of all available behaviors being able to operate over the shared environment.

Example 2.1.1. The notion of behavior is illustrated by services of online news providers. The four available behaviors in Fig. 2.2 have transitions labeled by actions that belong to the set $A = \{$*archive*, *publish*, *put_caption*, *translate*, *upload_photo*, *upload_video*, *write_story*$\}$. [1] Viewed as an available behavior, $\mathcal{B}_4$ has the capabilities to upload a photo, put a caption on the photo and translate the caption into English, and in that order. Notice that, the notion of environment is not used in this example. The assumption is that each behavior defines its actions independently from the others. The interleaving of $\mathcal{B}_1$ to $\mathcal{B}_4$ makes the system. ■

Example 2.1.2. The target behaviors in Fig. 2.3 represent the ways of doing news by traditionalist journalists and web video reporters, respectively. Typically, a journalist posts its stories to online newspapers through remote services (behaviors) that allow for writing a story text, translating the text into English, and archiving and publishing the story. A reporter issues requests to upload and archive a video, or upload a photo, put a caption on the photo, translate the caption into English, and then archive the photo. ■

In the case that behaviors cannot function in a standalone way, their real capabilities depend on both themselves and the environment operating on it. So, from this point, the notion of enacted behavior is defined.

Definition 2.1.3. Given a behavior $\mathcal{B}=\langle B, \delta, b_0, G, F \rangle$ and an environment $\mathcal{E} = \langle A, E, e_0, \rho \rangle$, the enacted behavior of $\mathcal{B}$ on $\mathcal{E}$ is the tuple $\mathcal{T}_\mathcal{B} = \langle S_\mathcal{B}, A, \delta_\mathcal{B}, s_{\mathcal{B}_0}, F_\mathcal{B} \rangle$,

1. A transition labeled with actions separated by a slash in $\mathcal{B}_1$ is just a more compact notation to represent multiple transitions. Each initial state is indicated by a dark arrow, and each final state is indicated by two concentric circles.

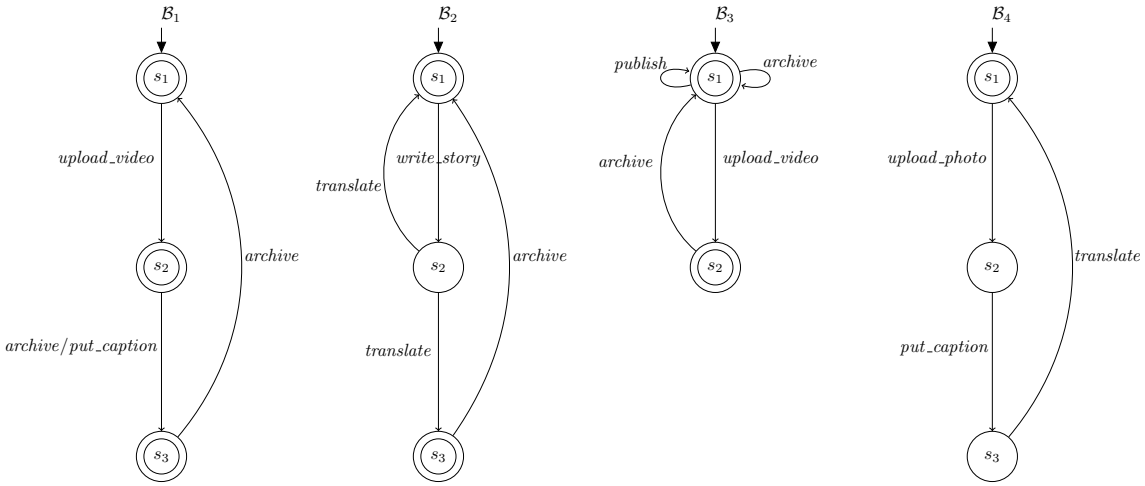


Figure 2.2: The available behaviors

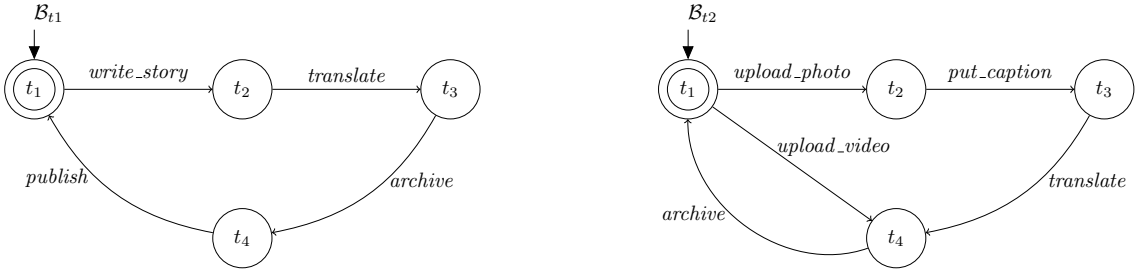


Figure 2.3: Two target behaviors

where $S_{\mathcal{B}} = B \times E$ is the (finite) set of states, A is the same set of actions as defined in $\mathcal{E}$, $\delta_{\mathcal{B}} \subseteq S_{\mathcal{B}} \times A \times S_{\mathcal{B}}$ is the transition relation, $s_{\mathcal{B}_0} = \langle b_0, e_0 \rangle \in S_{\mathcal{B}}$ is the initial state, and $F_{\mathcal{B}} = F \times E$ is the set of final states. The transition $\langle \langle b, e \rangle, a, \langle b', e' \rangle \rangle \in \delta_{\mathcal{B}}$ if and only if:

- $\langle e, a, e' \rangle \in \rho$;
- $\langle b, g, a, b' \rangle \in \delta$ and $g(e) = \top$.

It means that $\mathcal{B}$ and $\mathcal{E}$ synchronize on all actions.

Given a state $\boldsymbol{b} = \langle b, e \rangle \in S_{\mathcal{B}}$, b and e are denoted by $beh(\boldsymbol{b})$ and $env(\boldsymbol{b})$, respectively. As depicted in Fig. 2.1, the notion of enacted target behavior is defined from the target behavior on $\mathcal{E}$.

Definition 2.1.4. The enacted target behavior $\mathcal{T}_{\mathcal{B}_t}$ is the tuple $\langle S_{\mathcal{B}_t}, A, \delta_{\mathcal{B}_t}, s_{\mathcal{B}_{t0}}, F_{\mathcal{B}_t} \rangle$ such that $\mathcal{T}_{\mathcal{B}_t}$ is the enacted behavior of $\mathcal{B}_t$ on $\mathcal{E}$.

All available behaviors in a system operate in the shared environment in an interleaved fashion, called the enacted system behavior.

Definition 2.1.5. Given a system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$, the enacted system behavior of $\mathcal{S}$ is the tuple $\mathcal{T}_{\mathcal{S}} = \langle S_{\mathcal{S}}, A, I_n, \delta_{\mathcal{S}}, s_{\mathcal{S}0}, F_{\mathcal{S}} \rangle$, where $S_{\mathcal{S}} = B_1 \times \ldots \times B_n \times E$, $I_n = \{1, \ldots, n\}$ is the set of behavior indexes, $\delta_{\mathcal{S}} \subseteq S_{\mathcal{S}} \times A \times I_n \times S_{\mathcal{S}}$ is the transition relation, $s_{\mathcal{S}0} = \langle b_{10}, \ldots, b_{n0}, e_0 \rangle$ is the initial state, and $F_{\mathcal{S}} = \{\boldsymbol{s} \in S_{\mathcal{S}} | beh_i(\boldsymbol{s}) \in F_i \text{ for all } i \in I_n\}$ is the set of final states. The transition:

$$\langle \langle b_1, \ldots, b_i, \ldots, b_n, e \rangle, \langle a, i \rangle, \langle b_1, \ldots, b'_i, \ldots, b_n, e' \rangle \rangle \in \delta_{\mathcal{S}}$$

if and only if:

- $\langle e, a, e' \rangle \in \rho$;
- $\langle b_i, g_i, a, b'_i \rangle \in \delta_i$ and $g_i(e) = \top$ for all $i \in I_n$.

It means that the environment synchronizes with behavior $\mathcal{B}_i$ on action a independently of the other behaviors.

When there is no environment, the actions that belong to A are given out to available behaviors, that is, $\mathcal{B}_i = \langle B_i, A_i, \delta_i, b_{i0}, F_i \rangle$, where guards are eliminated and elements are defined as in Def. 2.1.2, but with $\delta_i \subseteq B_i \times A_i \times B_i$. In the same way $\mathcal{B}_t = \langle B_t, A_t, \delta_t, b_{t0}, F_t \rangle$, where elements are defined as in Def. 2.1.2, but with $\delta_t \subseteq B_t \times A_t \times B_t$ and $A_t \subseteq \cup_i A_i$. In that case, the notions of enacted system behavior and enacted target behavior are unnecessary. There are only the system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n \rangle$ and the target behavior $\mathcal{B}_t$. The system behavior, also denoted by $\mathcal{S}$, is the tuple $\mathcal{S} = \langle S, A, I_n, \delta, s_0, F \rangle$, where $S = B_1 \times \ldots \times B_n$, $A = \cup_i A_i$, $s_0 = \langle b_{10}, \ldots, b_{n0} \rangle$, $F = F_1 \times \ldots \times F_n$, and $\delta \subseteq S \times A \times I_n \times S$ is the system transition relation. The transition $\langle \langle b_1, \ldots, b_i, \ldots, b_n \rangle, \langle a, i \rangle, \langle b_1, \ldots, b'_i, \ldots, b_n \rangle \rangle \in \delta$ if and only if $\langle b_i, a, b'_i \rangle \in \delta_i$ for all $i \in I_n$.

2.2 Controller Synthesis

In the original behavior composition framework, there are two ways for synthesizing a controller generator, one way is based on an algorithm, which calculates the largest ND-simulation. The other concerns the calculation of a winning strategy of a corresponding two-player safety game by using the model checker SMV/TLV.

2.2.1 Synthesis Based on an ND-simulation Relation

Definition 2.2.1. Let $\boldsymbol{t} \in S_{\mathcal{B}_t}$ and $\boldsymbol{s} \in S_{\mathcal{S}}$, an ND-simulation relation of $\mathcal{T}_{\mathcal{B}_t}$ by $\mathcal{T}_{\mathcal{S}}$ is a relation $R \subseteq S_{\mathcal{B}_t} \times S_{\mathcal{S}}$, such that $\langle \boldsymbol{t}, \boldsymbol{s} \rangle \in R$ implies:

1. $env(\boldsymbol{t}\ ) = env(\boldsymbol{s})$;
2. if $\boldsymbol{t} \in F_{\mathcal{B}_t}$, then $\boldsymbol{s} \in F_{\mathcal{S}}$;
3. for all actions $a \in A$, there is a $k \in I_n$ such that for all transitions $\langle \boldsymbol{t}, a, \boldsymbol{t'} \rangle \in \delta_{\mathcal{B}_t}$:
   - there is a transition $\langle \boldsymbol{s}, \langle a, k \rangle, \boldsymbol{s'} \rangle \in \delta_{\mathcal{S}}$ with $env(\boldsymbol{t'}\ )$=$env(\boldsymbol{s'})$;
   - for all transitions $\langle \boldsymbol{s}, \langle a, k \rangle, \boldsymbol{s'} \rangle \in \delta_{\mathcal{S}}$ with $env(\boldsymbol{t'}\ )$=$env(\boldsymbol{s'})$, it is the case that $\langle \boldsymbol{t'}, \boldsymbol{s'} \rangle \in R$.

The symbol "$\preceq$" is used to denote that a state $\boldsymbol{t} \in S_{\mathcal{B}_t}$ is ND-simulated by a state $\boldsymbol{s} \in S_{\mathcal{S}}$, that is, there exists an ND-simulation relation R of $\mathcal{T}_{\mathcal{B}_t}$ by $\mathcal{T}_{\mathcal{S}}$ such that $\langle \boldsymbol{t}, \boldsymbol{s} \rangle \in R$.

The algorithm that computes the largest ND-simulation relation is presented in Algorithm 1. It eliminates iteratively states of $S_{\mathcal{B}_t} \times S_{\mathcal{S}}$ that do not satisfy the conditions of Def. 2.2.1.

The theorem 1 in [32] proves that a controller of $\mathcal{B}_t$ on $\mathcal{S}$ exists if and only if $s_{\mathcal{B}_{t0}} \preceq s_{\mathcal{S}0}$. From the largest ND-simulation relation, a finite state machine, called controller generator, can be derived. It is formally defined as follows.

Definition 2.2.2. A controller generator is the tuple $CG = \langle \Sigma, A, I_n, \xi, \omega \rangle$, where $\Sigma = \{\langle \boldsymbol{t}, \boldsymbol{s} \rangle \in S_{\mathcal{B}_t} \times S_{\mathcal{S}} | \boldsymbol{t} \preceq \boldsymbol{s}\}$ is the set of CG states. Given a state $\sigma = \langle \boldsymbol{t}, \boldsymbol{s} \rangle \in \Sigma$, $\boldsymbol{t}$ and $\boldsymbol{s}$ are denoted by $com_{\mathcal{B}_t}(\sigma)$ and $com_{\mathcal{S}}(\sigma)$, respectively, and $\xi \subseteq \Sigma \times A \times I_n \times \Sigma$ is the CG transition relation. The transition $\langle \sigma, \langle a, k \rangle, \sigma' \rangle \in \xi$ if and only if:

- $\langle com_{\mathcal{B}_t}(\sigma), a, com_{\mathcal{B}_t}(\sigma') \rangle \in \delta_{\mathcal{B}_t}$;

Algorithm 1 *LNDS*$(\mathcal{T}_{\mathcal{B}_t}, \mathcal{T}_{\mathcal{S}})$—Calculation of the largest ND-simulation

$R := S_{\mathcal{B}_t} \times S_{\mathcal{S}} \ \setminus \{\langle \boldsymbol{t}, \boldsymbol{s}\rangle \mid env(\boldsymbol{t}) \neq env(\boldsymbol{s}) \vee (\boldsymbol{t} \in F_{\mathcal{B}_t} \wedge \boldsymbol{s} \notin F_{\mathcal{S}})\}$;
repeat
  $R' := R$
  $R := R \setminus \{\langle \boldsymbol{t}, \boldsymbol{s}\rangle \mid (\exists\ \boldsymbol{t'} \in S_{\mathcal{B}_t}$ such that $\langle \boldsymbol{t}, a, \boldsymbol{t'}\rangle \in \delta_{\mathcal{B}_t}) \wedge$
    $(\neg\exists\ \boldsymbol{s'} \in S_{\mathcal{S}}$ such that $\langle \boldsymbol{s}, \langle a, k\rangle, \boldsymbol{s'}\rangle \in \delta_{\mathcal{S}} \wedge env(\boldsymbol{t'}) = env(\boldsymbol{s'}) \vee$
     $\exists\ \boldsymbol{s'} \in S_{\mathcal{S}}$ such that $\langle \boldsymbol{s}, \langle a, k\rangle, \boldsymbol{s'}\rangle \in \delta_{\mathcal{S}} \wedge env(\boldsymbol{t'}) = env(\boldsymbol{s'}) \wedge$
    $\langle \boldsymbol{t'}, \boldsymbol{s'}\rangle \notin R)\}$
until $R = R'$
return R.

– $\langle com_{\mathcal{S}}(\sigma), \langle a, k\rangle, com_{\mathcal{S}}(\sigma')\rangle \in \delta_{\mathcal{S}}$;

– for all $\langle com_{\mathcal{S}}(\sigma), \langle a, k\rangle, \boldsymbol{s'}\rangle \in \delta_{\mathcal{S}}$, $\langle com_{\mathcal{B}_t}(\sigma'), \boldsymbol{s'}\rangle \in \Sigma$.

The function $\omega : \Sigma \times A \to 2^{I_n}$ is an output function defined as:

$$\omega(\sigma, a) = \{k \mid \ \exists \sigma' \in \Sigma \text{ such that } \langle \sigma, \langle a, k\rangle, \sigma'\rangle \in \xi\}.$$

Given an action and current state of system, the output of CG is the set of available behaviors that execute the action while preserving the largest ND-simulation relation. Notice that, computing CG from the largest ND-simulation relation just involves checking local conditions [31, 32].

The notions of trace, history, and projection are introduced from a controller generator. A CG trace τ_{CG} is a sequence $\sigma^0 \xrightarrow{a^1,k^1} \sigma^1 \xrightarrow{a^2,k^2} \cdots$ with $\langle \sigma^{i-1}, \langle a^i, k^i\rangle, \sigma^i\rangle \in \xi$, for all $i > 0$, and a CG history is a finite prefix of a CG trace. Given a controller generator history h_{CG}, the last state of the history is denoted by $last(h_{CG})$. A projected system trace is defined as $proj_{\mathcal{S}}(\tau_{CG}) = com_{\mathcal{S}}(\sigma^0) \xrightarrow{a^1,k^1} com_{\mathcal{S}}(\sigma^1) \xrightarrow{a^2,k^2} \cdots$ and a projected target trace is a sequence of $proj_{\mathcal{B}_t}(\tau_{CG}) = com_{\mathcal{B}_t}(\sigma^0) \xrightarrow{a^1} com_{\mathcal{B}_t}(\sigma^1) \xrightarrow{a^2} \ldots$.

Definition 2.2.3. Let $\mathcal{H}_{CG}$ be a set of all controller generator histories, a selection function $CGP : \mathcal{H}_{CG} \times A \to I_n$ is defined from the output function ω such that for all $h_{CG} \in \mathcal{H}_{CG}$ and $a \in A$, $CGP(h_{CG}, a) \in \omega(last(h_{CG}), a)$, if $\omega(last(h_{CG}), a)$ is non-empty.

The function CGP selects one available behavior among those that are able to execute the current action with respect to the last state of a given history.

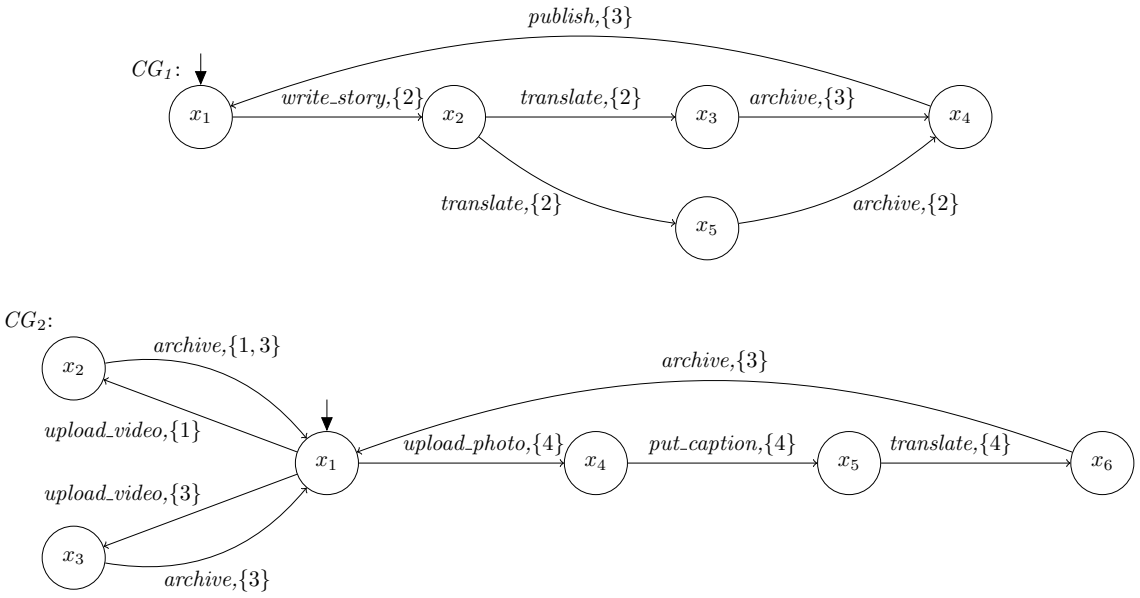


Figure 2.4: Two controller generators

A family of generated controllers, called also *compositions* of $\mathcal{B}_t$ on $\mathcal{S}$, can be derived from the controller generator. Notice that, in some cases, the number of generated controllers can be infinite [32].

Definition 2.2.4. A generated controller is a function $P : \mathcal{H} \times A \to I_n$ such that, for a given controller generator history h_{CG} and for each enacted system behavior history $h \in \mathcal{H}$ and action $a \in A$, if $h = proj_{\mathcal{S}}(h_{CG})$, then $P(h, a) = CGP(h_{CG}, a)$.

Example 2.2.1. Figure 2.4 shows the controller generator for each target behavior described in Example 2.1.2. There is only one possible generated controller from CG_1, but an infinity from CG_2 due to the presence of loops $x_1x_2x_1$ and $x_1x_3x_1$. When the generated controller inferred from CG_1 delegates the action *translate* to $\mathcal{B}_2$, its next state can be x_3 or x_5 according to the nondeterministic choice made by $\mathcal{B}_2$ to execute *translate* from its state s_2. Due to nondeterminism, the controller must observe the current state of the system after every execution of an action to determine its own next state. ■

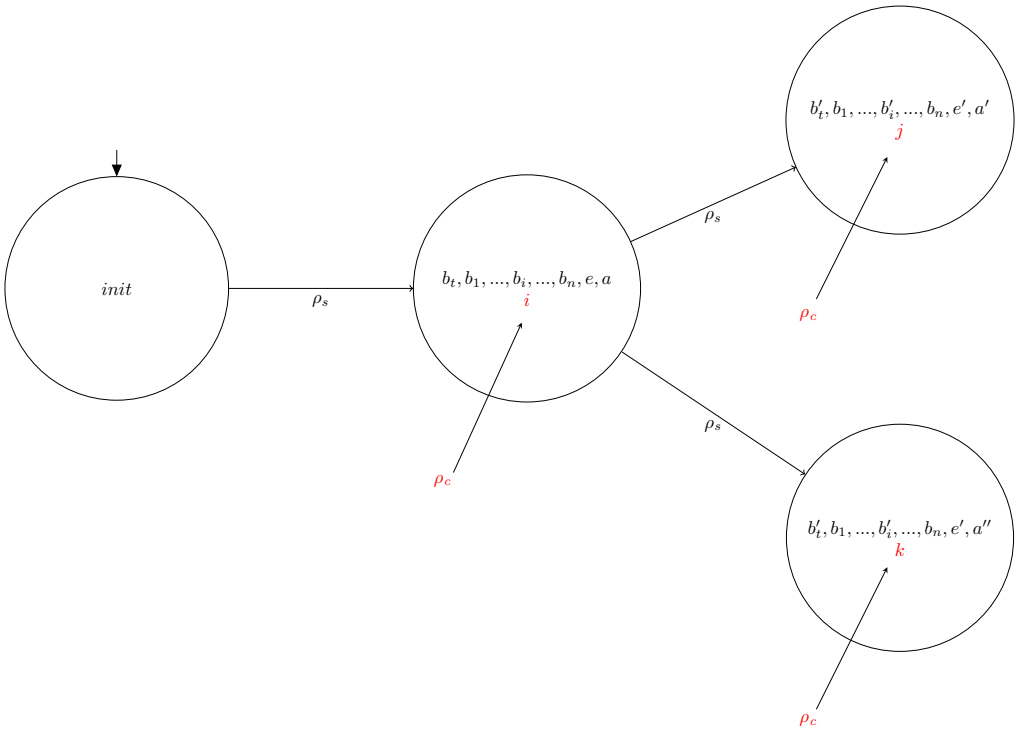


Figure 2.5: Two players of a safety game

2.2.2 Synthesis Based on a Safety-Game Structure

For the purpose of controller synthesis with the aid of an available model checking tool, the approach based on the largest ND-simulation is replaced by the calculation of a winning strategy of a corresponding two players in a safety game [32]. As depicted in Fig. 2.5, in such a game structure, one plays the role of the system and the other plays the role of the controller. The former keeps the information about the current state of the target behavior, available behaviors, and environment, and at each step, releases an action that must be executed. The latter returns an index indicating which available behavior in the system is able to perform the requested action. These data appear in two lines inside each state of the transition system in Fig. 2.5. The state *init* indicates an initial state. In this state, all available behaviors and the target behavior are in their initial states, the current action is empty and there is no controller reply, namely the index is zero.

```
MODULE main
 VAR
   sys: system Sys(ctr.index);
   ctr: system Ctr;
 DEFINE
   good := (ctr.initial & sys.initial) | !(sys.failure);

MODULE Ctr
 VAR
   index : 0..4;
 INIT
   index = 0
 TRANS
   case
    index=0 : next(index)!=0;
    index!=0 : next(index)!=0;
   esac
 DEFINE
   initial := (index=0);

MODULE Sys(index)
 VAR
  operation  : {start_op,archive,publish,put-caption,translate,upload-photo,upload-video,write-story};
  target : Target(operation);
  B1 : Behavior1(index,operation);
  B2 : Behavior2(index,operation);
  B3 : Behavior3(index,operation);
  B4 : Behavior4(index,operation);
 DEFINE
  initial := (B1.initial & B2.initial & B3.initial & B4.initial & target.initial & operation=start_op);
  failure := (B1.failure |B2.failure |B3.failure |B4.failure) |
             (target.final & !(B1.final & B2.final & B3.final & B4.final));
```

Figure 2.6: The main modules of a system and a controller in SMV

The transition relations ρ_s and ρ_c represent system moves and controller replies, respectively. More precisely, $\rho_s \subseteq X \times Y \times X$, where $X = B_t \times B_1 \times \ldots \times B_n \times E \times (A \cup \emptyset)$ and $Y = I_n \cup \{0\}$, and $\rho_c \subseteq X \times Y \times X \times Y$, where

$$\langle\langle b_t, b_1, \ldots, b_n, e, a\rangle, i, \langle b'_t, b'_1, \ldots, b'_n, e', a'\rangle, j\rangle \in \rho_c$$

if and only if $j \neq 0$. Notice that, $b_k = b'_k$ for all $k \in I_n \setminus \{i\}$.

The reader is referred to [32] for the detailed procedure that shows how to derive a safety-game structure from a behavior composition problem.

2.3 Implementation of Behavior Composition in SMV/TLV

Once behavior composition has been translated into the safety-game structure, it can be implemented with a model checking tool like TLV [50]. TLV (Temporal Logic Verifier) is a tool for the purpose of verification of LTL specifications. It uses Boolean Decision Diagrams (BDDs) for indicating state valuations and transitions. The inputs

```
MODULE Target(op)
 VAR
   state : {start_st,t1,t2,t3,t4};
 INIT
   state = start_st & op = start_op
 TRANS
  case
    state = start_st & op = start_op : next(state) = t1 & next(op) = write-story;
    state = t1  & op = write-story  : next(state) = t2 & next(op) in {translate} ;
    state = t2  & op = translate  : next(state) = t3 & next(op) = archive ;
    state = t3  & op = archive : next(state) = t4 & next(op) = publish ;
    state = t4  & op = publish : next(state) = t1 & next(op) = write-story ;
  esac
 DEFINE
   initial := state=start_st & op=start_op;
   final := state in {t1};

MODULE Behavior1(index,operation)
 VAR
   state : {start_st,a1,a2,a3};
 INIT
   state=start_st
 TRANS
   case
    state=start_st & operation=start_op & index=0: next(state)=a1;
    (index != 1) : next(state) = state;
    (state=a1 & operation = upload-video) : next(state) in {a2};
    (state=a2 & operation = archive) : next(state) in {a3};
    (state=a2 & operation = put-caption) : next(state) in {a3};
    (state=a3 & operation = archive) : next(state) in {a1};
   esac
 DEFINE
   initial := state=start_st & operation=start_op  & index = 0 ;
   failure := index = 1 & !( (state = a1 & operation in {upload-video}) |
            (state = a2 & operation in {archive,put-caption}) | (state = a3 & operation in {archive}));
             final := state in {a1,a2,a3};
```

Figure 2.7: The modules of a target behavior and an available behavior

of TLV are an LTL specification written in SMV and a synthesis procedure. The latter is based on the safety-game structure and its output represents a controller generator. SMV (Symbolic Model Verifier) is a symbolic model checking tool supporting the verification of temporal logic (LTL and CTL) properties of finite-state machines. In SMV, each element of the safety-game players, namely available behaviors, target behavior, and controller, is implemented as a module [32]. Figure 2.6 includes the main modules of services of online news providers introduced in Example 2.1.1. The module `main` contains two submodules: the controller `Ctr` and the system `Sys`. The former returns the index of an available behavior executing the requested action of the target behavior. The latter chooses the next action that must be executed. More specifically, the system module contains the actions (operations), the target behavior, and four available behaviors. Figure 2.7 gives the SMV modules of the target behavior $\mathcal{B}_{t1}$ and the available behavior $\mathcal{B}_1$. The transitions part (`TRANS`) of the target module indicates how the desired actions are released. Such actions are inputs in the available behavior module.

```
Check Realizability
Specification is realizable
Check that a symbolic strategy is correct
Transition relation is complete
All winning states satisfy invariant
Automaton States

State 1
 sys.operation = start_op    sys.target.state = start_st     sys.B1.state = start_st
 sys.B2.state = start_st     sys.B3.state = start_st       sys.B4.state = start_st
 ctr.index = 0

State 2
 sys.operation = write-story      sys.target.state = t1        sys.B1.state = a1
 sys.B2.state = b1      sys.B3.state = c1      sys.B4.state = d1
 ctr.index = 2

State 3
 sys.operation = translate     sys.target.state = t2       sys.B1.state = a1
 sys.B2.state = b2      sys.B3.state = c1      sys.B4.state = d1
 ctr.index = 2
...

Automaton Transitions
 From 1 to  2
 From 2 to  3
 From 3 to  4 5
 ...

Automaton has 6 states, and 7 transitions
user time: 0.016 s
BDD nodes allocated: 10037
max amount of BDD nodes allocated: 10037
Bytes allocated: 655424
```

Figure 2.8: The TLV output (controller generator)

Figure. 2.8 shows the TLV output of the example, namely the realized controller generator. The first part of the output indicates whether the target behavior $\mathcal{B}_{t1}$ is realized or not. Since it is realizable, the details about the automaton forming the controller generator are provided. It should be noted that, the synthesized controller generator is in the format of a safety-game structure. Each state in the structure encompasses the system move and controller reply. The former contains the current state of the target behavior `sys.target.state`, available behaviors `sys.Bi.state`, and the current action `sys.operation`. The latter contains the index of the available behavior handling the action, namely `ctr.index`. As depicted in Fig. 2.8, in the state `State1`, the controller generator does not select a behavior, `ctr.index=0`, as it is in the initial state.

Appendix B provides the details about all modules of the Example 2.1.1 along with the TLV outputs.

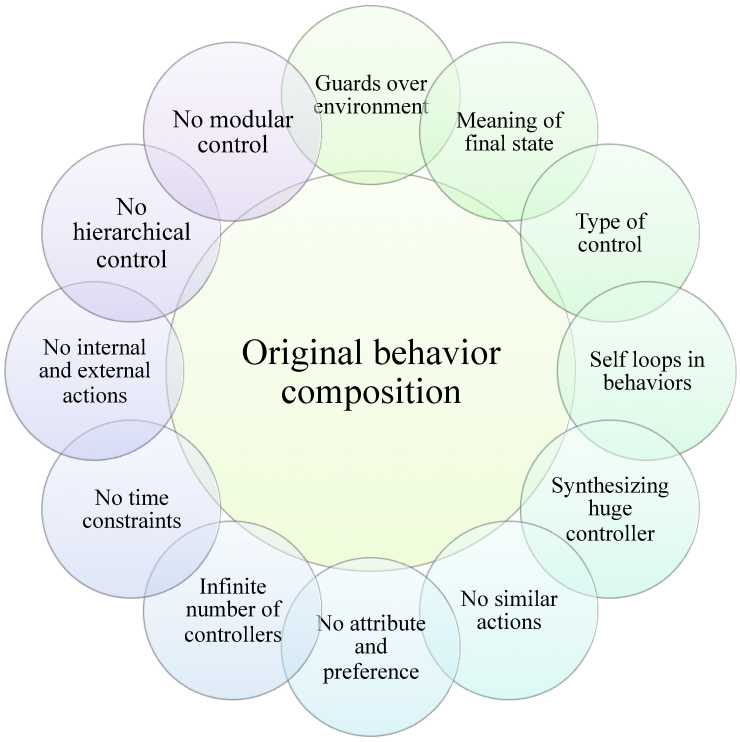


Figure 2.9: The drawbacks of original framework

2.4 Some Drawbacks in the Original Framework

The original behavior composition framework has several interesting features. For instance, the underlying synthesis technique is sound and complete (see theorem 3 in [32]). Furthermore, it has an acceptable computational complexity if the number of behaviors is not excessive (see theorem 2 in [32]). Though, as depicted in Fig. 2.9, the framework suffers from several drawbacks in real scenarios. Such drawbacks are enumerated in the following items.

– When the environment is explicit in a model, guards can be added on the transitions of the available behaviors. These guards express conditions on states of the environment and thus lead to some restrictions on the behaviors. The use of an explicit environment in a model is not very widespread in practice and in the scientific literature. The environment is only useful when one wants to impose local conditions on the triggering of transitions. However, the guards are

only considered in the calculation of the enacted system behavior from which controllers are synthesized.

– In the implementation of behavior composition through SMV/TLV, even if an available behavior does not handle any action of a target behavior, its initial state must be a final state for the aim of controller synthesis. This is due to the condition when the target behavior is in a final state: all available behaviors must be in their final states (see Def. 2.2.1). So, the role of final states in the available behaviors is not clear in the original framework. For example, for the realization of the target behavior $\mathcal{B}_{t1}$ in Example 2.1.2, even if the actions of $\mathcal{B}_{t1}$ are not delegated to the behaviors $\mathcal{B}_1$ and $\mathcal{B}_4$, their initial states must be final. Otherwise, no controller generator can be synthesized, in particular CG_1 depicted in Fig. 2.4.

– Both state-based and language-based control are simultaneously prevalent in the original framework. More precisely, in Def. 2.2.3, the output function ω exhibits a state-based control, while in Def. 2.2.4, the function P corresponds to a language-based control. Due to the nondeterminism of available behaviors, a generated controller must necessarily observe the current state of the system and makes a decision based on both the observed state and the action that must be executed. However, in the original framework, the reason of adopting a language-based control is not really justified. In fact, a control based on histories is not relevant when the environment and available behaviors are nondeterministic.

– When the original behavior composition problem is translated into the safety-game structure implemented in SMV/TLV, there must be self loops or loops in both target and available behaviors to support an infinite play between the system and the controller (see lemma 10 in [32]). Such an infinite play may bring out some restrictions for synthesizing a controller. In Example 2.2.1, assuming that both $\mathcal{B}_{t1}$ and $\mathcal{B}_{t2}$ have no loops, and their operations must be executed only once, although a controller generator exists for each target behavior based on a largest ND-simulation relation, there is no solution in the synthesis approach based on the safety-game structure unless dummy self loops are added in these two target behaviors.

– In the original framework, a huge transition system (controller generator) is synthesized. Then, a smaller transition system (generated controller) is extracted from the controller generator. However, such a method is time and memory consuming.
– In Algorithm 1, there must be an exact match between the actions of the target behavior and those of available behaviors. To apply the behavior composition framework in the context of semantic-based systems, analogous to the semantic web, actions offered by an available behavior (service broker) and similar to the ones requested by the target behavior should be matched. The actions having different names but the same functionality are considered similar. For instance, in Fig. 2.3, providing that the target behavior $\mathcal{B}_{t2}$ demands *photo_title* instead of *put_caption*, these actions can be considered similar. So, the action *photo_title* can be delegated to the behavior $\mathcal{B}_4$.
– The actions handled by the available behaviors do not have any attribute. Furthermore, possible preferences expressed by a target behavior are not supported in the original framework. For example, in Fig. 2.2, the action *translate* can have an attribute *language* through which available behaviors, namely $\mathcal{B}_2$ and $\mathcal{B}_4$, may assign a set of languages to this attribute based on their capabilities. Moreover, through the target behavior, a journalist may specify a particular language as a preference for the action *translate*.
– In some cases, the number of generated controllers that can be extracted from a controller generator is infinite and the selection of a generated controller is often done arbitrary, namely decided by an oracle. However, given utility values or costs that can be attached to the transitions of a controller generator, a problem that can be raised is how the best or optimal solutions can be determined.
– The original behavior composition framework cannot provide a solution for real-time scenarios. More specifically, in such scenarios, the specification may include a set of deadlines for the execution of requested actions. Likewise, available behaviors may be subject to a number of time constraints for the execution of their actions. Several verifications must be done on the generated controllers to check whether the time constraints imposed by the target behavior can be satisfied with respect to those of the available behaviors.

– All actions of a target behavior must be sent to a generated controller, which delegates it to an appropriate available behavior (see Def. 2.2.1). This is very restrictive. Thus, two types of action, namely internal and external, can be taken into consideration in the implementation of a component. Contrary to an external action, an internal action is not sent to the generated controller. It is executed internally by the implementation.

– The original behavior composition framework has not been conceived to be used in the perspective of component-based software engineering. Therefore, behaviors have no relationship between each other, as those of the interface and implementation of a component. In fact, the interface is an abstraction of the implementation, which hides some of its actions. Such a capability can give rise to the provision of hierarchical composition.

– Modular control was not studied in the original behavior composition framework. However, if the controller, already synthesized for a given target behavior, be reusable, then the notion of modular control naturally appears. Instead of synthesizing a new controller for a new target behavior, precomputed controllers are reused to realize the new target. For instance, assuming that a new complex target behavior is expressed as the combination of several plain target behaviors, where for each plain target, there exists a precomputed controller. So, the combination of the precomputed controllers can realize the complex target.

Chapter 3

From Synthesis to Simulation and Verification

In the behavior composition framework, once an orchestrator has been synthesized for a target behavior, several interactions are done among the target behavior, the orchestrator, and available behaviors. If both the target behavior and available behaviors are subject to some constraints (other than operation ordering and non-blocking) for the execution of operations, a question that can be raised is how such interactions can be simulated through a model checking tool to verify whether the orchestrator satisfies the constraints.

To show the importance of the simulation and verification of such interactions in a real scenario, this chapter integrates the behavior composition framework into the service-oriented IoT architecture presented in Sect. 1.4, in which services are atomic and applications are real-time.[1] In this architecture, a composite service interacts with a set of available atomic services (available behaviors) through its orchestrator. Such interactions form a closed-loop control system.

Due to the characteristics of the real-time IoT applications, the behavior composition framework requires a revision in order to be integrated into the aforementioned architecture. More specifically, these applications constitute time-critical reactive systems that are subject to time constraints. Given such a characteristic, this chapter revises the behavior composition framework and provides suggestions for the simulation and verification of the closed-loop control system through the steps summarized as follows.

– The introduction of a real-time version of the behavior composition framework, in which timed automata are used instead of untimed transition structures for modeling and verifying behaviors. First, the orchestrator synthesis is done by disregarding the time constraints. Then, the latter are verified.

1. The framework can be also integrated into other architectures in which services are not atomic.

– The development of a scenario in the form of a sequence diagram to show the interactions between a composite service under control and available atomic services.

– The simulation of the closed-loop control system in UPPAAL and its verification under some conditions written in a subset of a temporal logic to detect possible deadlocks.

– The introduction of two different scenarios for a composite service and, for each scenario, the verification of the composite service with respect to some real-time requirements.

3.1 Real-Time Behavior Composition Framework

One of the elements of a timed automaton is a set of clocks C, which are nonnegative real valued variables. A conjunctive formula of terms in the form of $x \sim c$ or $y - x \sim c$, where $x, y \in C$, $c \in \mathbb{N}$, and $\sim \in \{\leq, <, =, >, \geq\}$, is used to express a clock constraint [2]. A behavior is specified by a timed automaton $\langle Q, A, \eta, q_0, F, C, H \rangle$, where Q is a finite set of states (also called locations); A is a finite set of operations (also called actions); $\eta \subseteq Q \times G(C) \times A \times 2^C \times Q$ is the transition relation, with $G(C)$ the set of constraints over C (also called guards); $q_0 \in Q$ is the initial state; $F \subseteq Q$ is the set of final states; C is a finite set of clocks; and $H : Q \rightarrow G(C)$ is a function, which assigns an invariant to every state. The latter indicates the amount of time that may be spent in a state. The tuple $\langle q, g, a, D, q' \rangle$ represents a transition from q to q' on operation a under the clock constraint g. The clocks that belong to $D \subseteq C$ are reset to zero when the transition is taken. [2]

Given the timed model of a behavior, an automaton $\mathcal{B}_t = \langle Q_t, A, \eta_t, q_{t0}, F_t, C_t, H_t \rangle$ is the target service (or the implementation of the composite service under construction) and automata $\mathcal{B}_i = \langle Q_i, A_i, \eta_i, q_{i0}, F_i, C_i, H_i \rangle$, for all $i \in I_n = \{1, \ldots, n\}$, are available services (more specifically their interface). Furthermore, the system is $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n \rangle$.

When the guards and clock reset on transitions of the target and available services

2. Though clock constraints are only over transitions in [2], such constraints are also considered over states in other related manuscripts to denote the time lapse on a state in a timed automaton.

are ignored, the modeling formalism reduces to the one of the original framework and the theoretical apparatus can be applied. However, both the largest ND-simulation relation and orchestrator generator defined in Sect. 2.2 require a revision, since the environment is omitted. The elimination of environment keeps the soundness of the largest ND-simulation algorithm, since it simply eases available behaviors to execute their operations without any guards.

An ND-simulation relation of $\mathcal{B}_t$ by $\mathcal{S}$ is a relation $R \subseteq B_t \times S$ such that $\langle t, s\rangle \in R$ implies:

Revision of Def. 2.2.1

1. if $t \in F_t$, then $s \in F$;
2. for all transitions $\langle t, a, t'\rangle \in \delta_t$:
   - there exists a transition $\langle s, a, k, s'\rangle \in \delta$;
   - for all transitions $\langle s, a, k, s'\rangle \in \delta$, $\langle t', s'\rangle \in R$.

Based on the revision of the largest ND-simulation relation, the orchestrator generator OG of $\mathcal{B}_t$ on $\mathcal{S}$ is $\langle \Sigma, A, I_n, \xi, \omega\rangle$, where: [3]

Revision of Def. 2.2.2

1. $\Sigma = \{\langle t, s\rangle \in B_t \times S \mid t \preceq s\}$ is the set of OG states made by all pairs of $\mathcal{B}_t$ and $\mathcal{S}$ states that belong to the largest ND-simulation relation;
2. ξ is the transition relation, where $\sigma \xrightarrow{a,k} \sigma'$ in ξ, if and only if:
   - there is a transition $t \xrightarrow{a} t'$ in $\mathcal{B}_t$;
   - there is a transition $s \xrightarrow{a,k} s'$ in $\mathcal{S}$;
   - for all $\langle t'', s''\rangle \in B_t \times S$, such that $s \xrightarrow{a,k} s''$ in $\mathcal{S}$ and $t \xrightarrow{a} t''$ in $\mathcal{B}_t$, it is the case that $\langle t'', s''\rangle \in \Sigma$;
3. $\omega : \Sigma \times A \to 2^{I_n}$ is the output function with $\omega(\sigma, a) = \{k \mid \exists \sigma' \in \Sigma : \sigma \xrightarrow{a,k} \sigma' \in \xi\}$.

The next step focuses on checking whether the orchestrators extracted from the

3. The notations $\langle t, a, t'\rangle$ and $\langle s, a, k, s'\rangle$ are equivalent to $t \xrightarrow{a} t'$ and $s \xrightarrow{a,k} s'$, respectively.

synthesized orchestrator generator satisfy the time constraints of both the target behavior and available behaviors. By doing so, several verifications should be done on the orchestrators through UPPAAL.

3.1.1 A Simple Illustrative Example

Consider a patient whose blood glucose level is monitored and controlled remotely by two nurses. The patient has been prepared with the necessary equipment, namely two IoT medical devices: a smart insulin pump and a blood-pressure monitor. A nurse in a mobile workstation is informed about the patient's status by a safety alarm device. Likewise, another nurse in a control room of a hospital takes care of the patient through her workstation. The graphical user interface comprises two touch buttons to start and stop the patient's medical care remotely, and one touch button to get his health status.

Figure 3.1 shows three timed automata (available behaviors), one for each medical device. They model device behaviors. The insulin pump can automatically test the blood-glucose level and inject a low or high dose of insulin. It can be reset and filled with insulin. Based on the patient's blood pressure, the blood-pressure monitor triggers an emergency call or resets the device. Finally, the patient safety alarm has three lamps, which roughly indicate his current health status. Just like the blood-pressure monitor, it can make an emergency call. The time constraints are imposed on the operations of these devices and each device has own local clocks. Assuming that, the time unit is second. From state p_1, the insulin pump takes between 2 minutes and 2 minutes and half to be filled. When the pump is full, it can test the glucose level of the patient between 20 and 30 seconds. Following that, the pump performs one of these operations based on the test result: inject a high dose of insulin, inject a low dose of insulin, or reset. The pump is able to inject a high dose of insulin between 30 and 50 seconds, inject a low dose between 30 and 45 seconds, and reset between 30 and 35 seconds. The blood pressure monitor is equipped by an arm cuff and a wrist cuff. From state b_1, the blood-pressure monitor tests the blood pressure of the patient through the arm cuff between 60 and 70 seconds. From state b_2, the blood pressure test can be performed faster by the wrist cuff between 20 and 25 seconds. The blood pressure monitor can repeat the pressure test through the wrist cuff even if

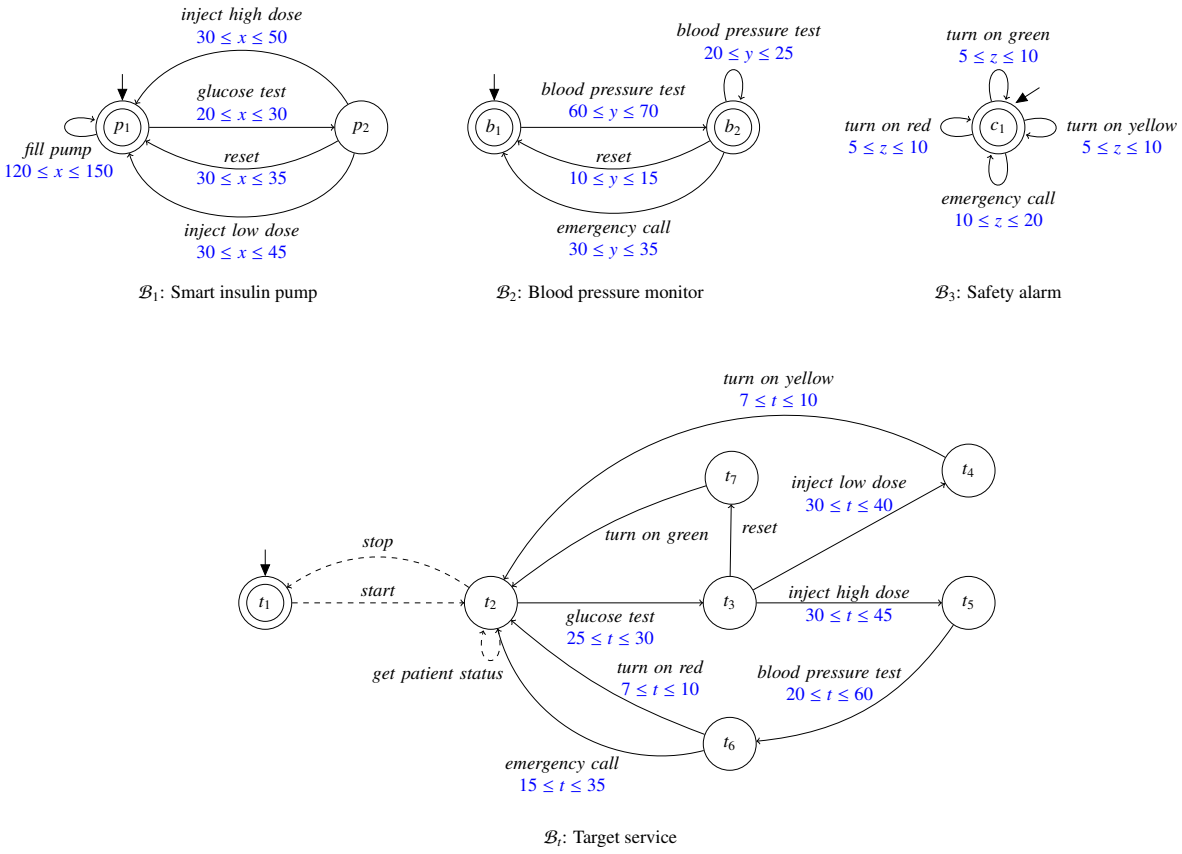


Figure 3.1: An IoT-based monitoring system for diabetes control

the monitor is not reset. If the test result be abnormal, the blood pressure test makes an emergency call between 30 and 35 seconds. Finally, the blood pressure monitor can be reset between 10 and 15 seconds. Given the safety alarm, each lamp can be turned on between 5 and 10 seconds. Besides, the alarm makes an emergency call between 10 and 20 seconds.

The target service (behavior) expresses requirements about the constrained behavior of a new service for the treatment of a diabetic patient. It is formally described by a timed automaton, also depicted in Fig. 3.1. This service must be realized with the aid of an orchestrator. The dashed transitions are labeled with operations, which are not delegated to available behaviors by the orchestrator. In this example, they correspond to the touch buttons of the nurse workstation user interface. The other

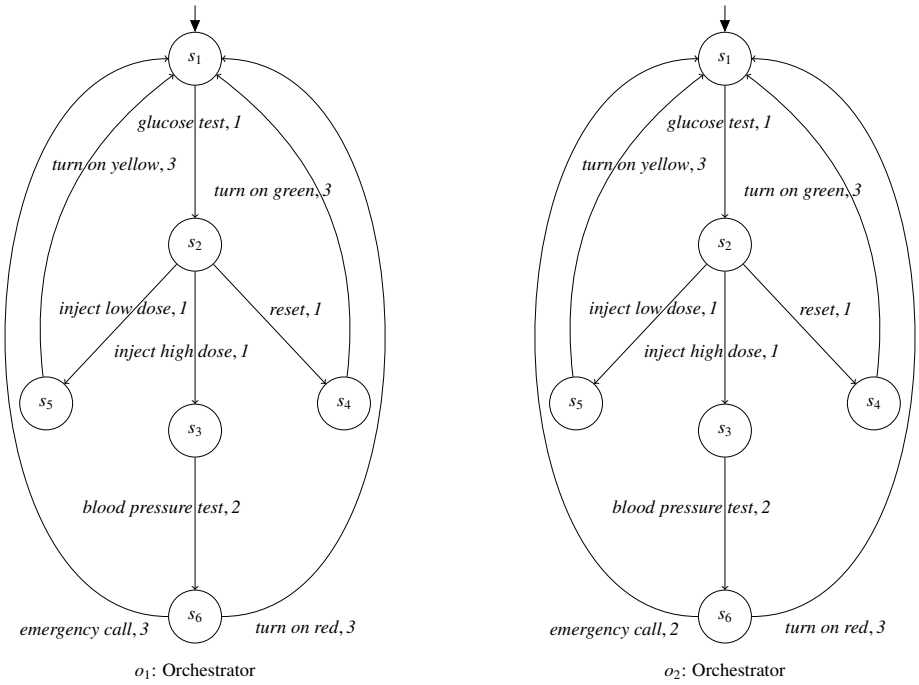


Figure 3.2: The possible orchestrators

transitions prescribe ordering constraints on the operations to be delegated to available behaviors. There are also time constraints, which represent restrictions imposed on response time from the user's point of view. For instance, the operation *inject high dose* must be executed between 30 and 45 seconds, or an emergency call must be made between 15 and 35 seconds. The transitions labeled with operations form cycles that are repeated until the nurse touches the stop button. The overall time constraints on a cycle should be weaker than those enforced by all available behaviors involved in a candidate composition.

It should be noted that the time constraints over clocks of timed automata do not necessarily reflect the reality. They have been set with the aim of explaining some aspects of the verification process.

After the implementation of the example in SMV/TLV, two orchestrators have been identified for realizing the target service. They are depicted in Fig. 3.2. One of them (o_1) delegates the *emergency call* to the safety alarm device. The other (o_2)

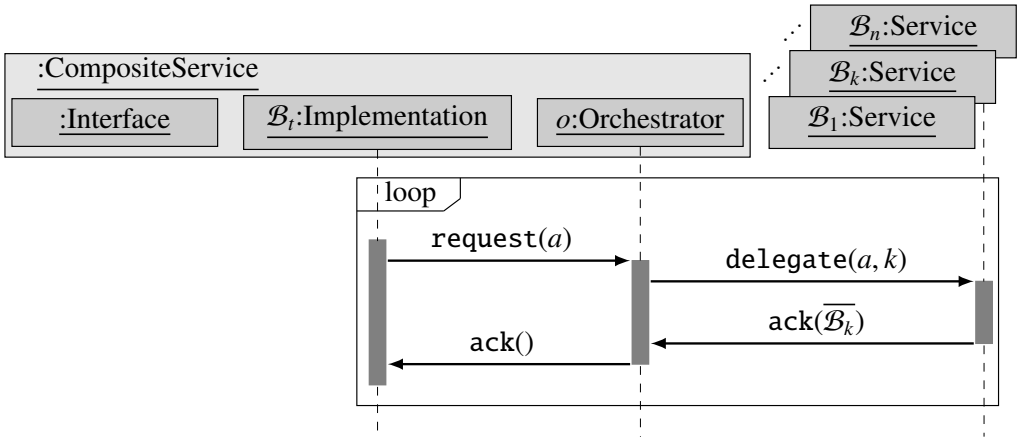


Figure 3.3: A closed-loop system

delegates it to the blood-pressure monitor.

Appendix C.1 provides the implementation of the example in SMV/TLV.

3.2 A Closed-Loop Control System

The orchestrator obtained from the synthesis procedure realizes the target service in the sense that it delegates its operations to the appropriate available atomic services (available behaviors), but without any guarantee about response delays prescribed by the real-time requirements. In order to remedy this situation, it is necessary to integrate the synthesized orchestrator into a closed-loop control system. The simulation and verification of composite services become then feasible.

The execution scenario, depicted in Fig. 3.3 in the form of a sequence diagram, shows the closed-control loop (or synchronization) between a composite service and a system formed from available atomic services. This diagram is in accordance with the original behavior composition framework [32]. More precisely, it focuses on the message exchanges between the implementation of the composite service ($\mathcal{B}_t$) and the interfaces of atomic services (available behaviors $\mathcal{B}_1$ to $\mathcal{B}_n$) of the system through the orchestrator of the composite service, which acts as a proxy redirecting requests and acknowledgments. Notably, each composite service consists of an interface, an implementation, and an orchestrator (see Sect. 1.3). The implementation attempts

to access an atomic service, which is available to perform the operation a, by sending the message $\texttt{request}(a)$ to the orchestrator o. Based on the system's current state and a control law, the latter chooses the k-th atomic service and forwards a to the selected atomic service $\mathcal{B}_k$ for execution by means of the message $\texttt{delegate}(a, k)$. Subsequent return messages are added to complete an entire cycle embedded in a loop frame element. The acknowledge message $\texttt{ack}(\overline{\mathcal{B}_k})$ returned by $\mathcal{B}_k$ contains its current state reached after the execution of a. It allows the orchestrator to determine the system state and adjust its own current state for the next execution cycle. Finally, the message $\texttt{ack}$ signals the completion of the operation to the composite service.

3.2.1 Implementation of the Closed-Loop Control System in Uppaal

The closed-loop control system depicted in Fig. 3.3 was implemented in Uppaal. The choice of this model checker came naturally, since the timed automaton formalism is intrinsic to Uppaal [13]. In the implementation of the closed-loop control system, there should be a distinction between deterministic and nondeterministic scenarios.

Deterministic Scenario

The exchanges of messages between a target behavior and available behaviors take place through synchronization channels. A channel for an operation a is declared as "$\texttt{chan}\ a$". The handshaking between the sender and receiver through the channel defined for a is denoted by "$a\texttt{!}$" on a transition of the sending automaton and "$a\texttt{?}$" on a transition of the receiving automaton. Two channels are necessary for an operation considering the sequence diagram in Fig. 3.3: "$\texttt{chan}\ a$" and "$\texttt{chan}\ ak$". The first is used by the implementation to send a request ($a\texttt{!}$), which is received by the orchestrator ($a\texttt{?}$). The second is used by the orchestrator that delegates the operation to the k-th available behavior ($ak\texttt{!}$), which accepts it for execution ($ak\texttt{?}$). This schema works well when the automata are deterministic. Furthermore, the acknowledgments can be ignored because there is at most one possible next state known by the orchestrator.

Example 3.2.1. The composite service for the treatment of diabetic patients described in Fig. 3.1 has been implemented in Uppaal in two versions with respect

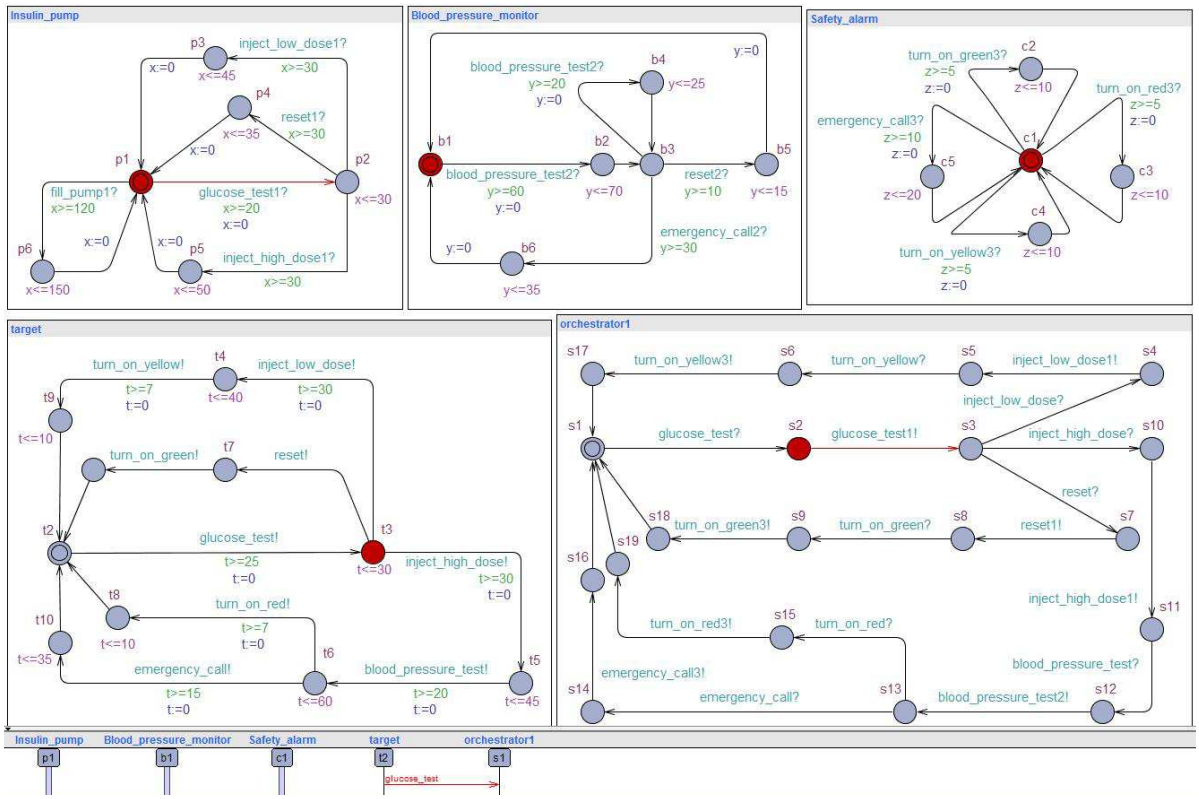


Figure 3.4: A view of the system in UPPAAL

to the two synthesized orchestrators provided in Fig. 3.2 and obtained from the orchestrator generator automatically calculated by TLV. Several simulations have been done in manual mode (i.e., step-by-step) to validate the models and automatic mode (i.e., a continuous way) to quickly identify deadlocks. Figure 3.4 shows a screenshot of a simulation in which the current state of each automaton appears in red and the interaction between the target service and orchestrator on the request `glucose_test` is displayed in the subwindow at the bottom of the screen. ■

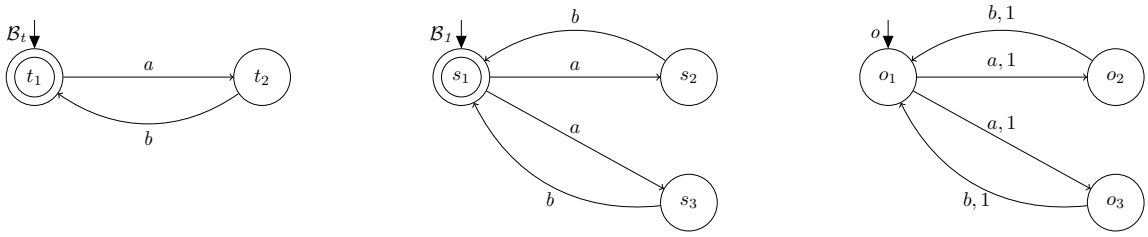
Each available behavior should have its local clock. In this end, a clock is declared as "`clock x`" in the declaration part of the template related to a behavior. The clocks of behaviors are continuously progress during the execution of operations in the closed-loop control system. In fact, a fraction of time that must be passed for taking a transition in a behavior impinges on the clocks of the other behaviors and

leads to a time progress in them. Given the simulation of the closed-loop control system in UPPAAL, it is impossible to stop such a time progress in the clock of a specific behavior.

Let the interval $[l_1,l_2]$, where l_1 and l_2 are integer numbers, be the time interval for performing an operation a in behavior $\mathcal{B}_k$ with the clock x. Assuming that, when the operation is executed in $\mathcal{B}_k$, the behavior reaches a new state s'. In order to implement such time interval in UPPAAL, a time constraint $x \geq l_1$ should be located on the transition related to the operation "$ak?$" and an invariant $x \leq l_2$ should be located on the state s' in $\mathcal{B}_k$. Figure 3.4 shows how time constraints in the available devices depicted in Fig. 3.1 were implemented in UPPAAL. It can be seen from Fig. 3.4 that some new states and dummy transitions have been added in the implementation of automata depicted in Fig. 3.1 to take into account the time intervals and clearly represent the reachable states.

Nondeterministic Scenario

A more complex schema is implemented in the presence of nondeterminism. Assuming that an orchestrator, for the realization of a target behavior, delegates an operation a to an available behavior $\mathcal{B}_k$, and this behavior has nondeterministic transitions for the execution of operation a. For the aim of implementing such situation in UPPAAL, for each state, which is directly reachable through a nondeterministic transition in $\mathcal{B}_k$, a new channel "`chan` aks" is declared. Such channels enable the available behavior to inform the orchestrator about its current state after the execution of a nondeterministic transition. By doing so, in the behavior $\mathcal{B}_k$, every nondeterministic transition labeled by "$ak?$" from s to s' is replaced by a transition labeled by "$ak?$" from s to s'' and a transition labeled "$aks!$" for s'' to s', where s'' is an intermediate new state, to send the current state of the behavior, namely s', to the orchestrator. Likewise, a similar construction is done in the orchestrator for each transition labeled by "$aks!$" in $\mathcal{B}_k$. A new transition with the label of "$aks?$" is inserted (with an intermediate state) in the orchestrator. Once a message "$ak!$ has been sent to $\mathcal{B}_k$, the orchestrator waits to receive the current state of $\mathcal{B}_k$ via the channel "$aks?$".



(a) Target behavior $\mathcal{B}_t$ (b) Available behavior $\mathcal{B}_1$ (c) Orchestrator o

Figure 3.5: A simple nondeterministic scenario

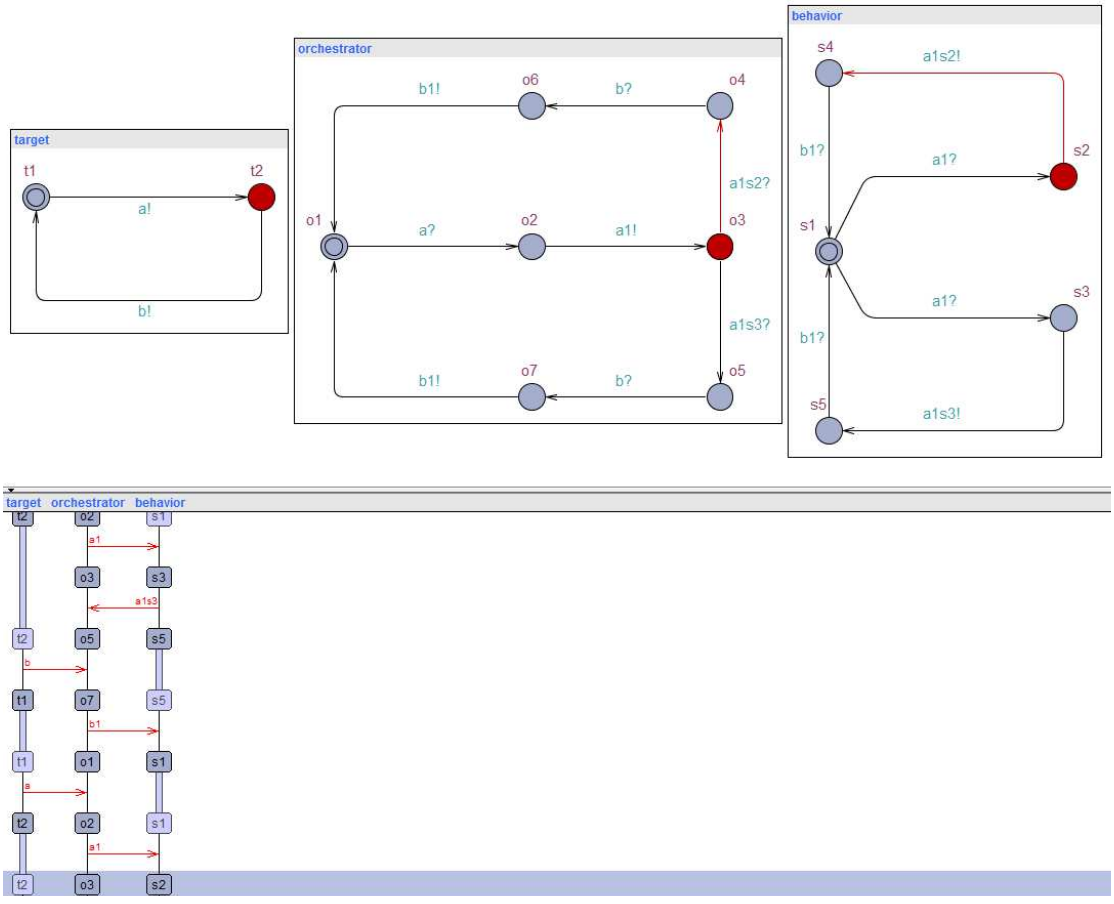


Figure 3.6: The simple nondeterministic scenario in UPPAAL

Example 3.2.2. Figure 3.5 shows a target behavior, an available behavior, and a synthesized orchestrator for the target. There are nondeterministic transitions in both the available behavior and orchestrator. The implementation of the example in UPPAAL is illustrated in Fig. 3.6. After the execution of a in $\mathcal{B}_1$, the orchestrator can be informed about the current state of the behavior through the handshaking between the channel "$a1s2!$" and "$a1s2?$" or between "$a1s3!$" and "$a1s3?$".

The execution of a nondeterministic transition is randomly selected in UPPAAL. The sequence diagram in the bottom of Fig. 3.6 indicates a part of the interactions after the twice executions of the closed-loop control system. It can be seen that, in the first execution, the transition $\langle s_1, a1?, s_3 \rangle$ is taken in $\mathcal{B}_1$, while in the second execution, the transition $\langle s_1, a1?, s_2 \rangle$ is taken. ■

3.3 Verification of Composite Services

The time constraints in both target service and available services were ignored during the synthesis of an orchestrator. Given the time constraints, when the closed-loop control is simulated, the reachability of all states in such an orchestrator is not guarantee. Since the orchestrator may be subject to deadlock from some states, where the time constraints in the target service and available atomic services are not satisfied. In this end, before construction of a composite service, several verifications must be done to detect such possible deadlocks. After a review of types of deadlocks and their classification, some verification formulas are proposed for the purpose of checking deadlock in a composite service.

3.3.1 Deadlock

A deadlock is a state where the system cannot progress more. A deadlock occurs from a state in a timed automaton when a transition cannot be taken due to the time progress or there is not any outgoing transition (i.e., the state is a terminal state). Generally, deadlocks are classified as *pure-actionlocks*, *time-actionlocks*, and *zeno-timelocks*, which are defined below.

Definition 3.3.1. [19] A pure-actionlock is a state where the system cannot execute any action transition, but time is permitted to progress. Let $\langle Q, A, \eta, q_0, F, C, H \rangle$ be

a timed automata, a state $q \in Q$ is a pure-actionlock if:

$$\forall d \in \mathbb{R}^{\geq 0}, (q+d) \in Q \text{ and } \nexists a \in A \text{ such that } (q+d) \xrightarrow{a} .$$

Definition 3.3.2. [19] A time-actionlock is a state where neither action nor time transitions can be executed. Let $\langle Q, A, \eta, q_0, F, C, H \rangle$ be a timed automaton, a state $q \in Q$ is a time-actionlock if:

$$\nexists a \in A, d \in \mathbb{R}^{>0}, \text{ such that } q \xrightarrow{a} \text{ or } q \xrightarrow{d} .$$

Definition 3.3.3. [19] A zeno-timelock is a state where system can perform transitions (actions or delays), but time cannot progress beyond a certain point. It shows a situation where the system executes an infinite number of actions in a finite period of time.

Notice that, in UPPAAL, there is no distinction among such a classification and they are just represented by the word `deadlock`.

3.3.2 Formulas for Checking Deadlock

Given the classification of deadlocks, the pure-actionlock is not the case in the closed-loop control system due to the infinite plays of system and orchestrator (controller) in the original framework of behavior composition (see Sect. 2.2.2). Besides, the assumption is that taking a transition in a target service or available atomic services leads to a time progress. Hence, the zeno-timelock is not a concern with this assumption, and the only type of deadlock that should be detected is time-actionlock.

The heart of interactions in a composite service is the orchestrator. Generally, when the closed-loop control is simulated in UPPAAL, one of the following situations may occur after the execution of a transition in the orchestrator:

- all states are reachable without any time-actionlock;
- all states are reachable but from some states, there possibly exist time-actionlocks;
- time-actionlock.

In order to check such situations, two different formulas can be proposed. The satisfaction of the following formula corresponds the first situation in a composite

service:

```
A[] not deadlock
```
(3.1)

If the formula is not satisfied, it means that there can be time-actionlock. By doing so, in order to detect the second or third situation, the reachability in the orchestrator can be tested through the following formula:

```
E<> not deadlock and orchestrator.s
```
(3.2)

This formula should be checked on each state s of the orchestrator. If the formula is satisfied for all states, it means the second situation. Otherwise, it signifies the third situation.

In some cases, a composite service, which involves an orchestrator with the second situation, can be utilized by some users who are optimistic about the reachability of all states in the orchestrator and does not have a concern about the possible time-actionlocks that can be occurred from some states. As a verification, the following formula enables such users to detect whether there exists a feasible time-actionlock in a given state s of the orchestrator.

```
E<> deadlock and orchestrator.s
```
(3.3)

The satisfaction of this formula represents a possible time-actionlock in the given state s.

Example 3.3.1. In order to check whether the orchestrators depicted in Fig. 3.2 have time-actionlock, the closed-loop control systems involving such orchestrators were tested by using Formula 3.1. The one involving orchestrator o_1 satisfied the formula, but the other, which involves o_2, did not satisfy. In the next step, the orchestrator o_2 was tested by using Formula 3.2 to check its reachability, and the formula was satisfied for all states in o_2. For detecting the time-actionlock in the orchestrator o_2, it was tested under Formula 3.3 and the following verification was satisfied.

```
E<> deadlock and orchestrator2.s14
```

It means that there exists a possible time-actionlock from s_{14}. Because the time of clocks continuously progress, it is possible that the clock of blood pressure monitor,

namely y, reaches a time greater than 35 seconds in state b_3. In this case, the orchestrator cannot delegate the operation *emergency call* to the device and a time-actionlock occurs. ■

Example 3.3.2. Assuming that, in the blood pressure monitor, there exists an invariant $y < 30$ in state b_3. In this case, the Formula 3.1 were not satisfied in both the closed-loop control systems involving the orchestrators o_1 and o_2, respectively. Because it is possible that the clock in state b_3 reaches a time greater than 25 seconds. So, the transition `blood_pressure_test2?` cannot be taken from the state.

Under the same assumption, they were individually tested by using Formula 3.2 in order to check the reachability of states in the orchestrators. The orchestrator o_1 satisfied the formula for all of its states. However, the following formula was not satisfied in o_2.

```
E<> not deadlock and orchestrator2.s14
```

Since the clock of monitor in state b_3 cannot reach a time exactly or greater than 30 seconds, while the time invariant for performing the operation is between 30 and 35 seconds. Then, the orchestrator o_2 cannot delegate the operation emergency call to the blood pressure monitor. ■

3.4 Real-Time User Requirements

Based on the time constraints in available atomic services, two different scenarios can be introduced for the construction of a composite service. In one scenario, the implementation of a composite service is in the form of a timed automaton and imposes the real-time requirements of the users in advance. In this scenario, the necessary condition for constructing a composite service is the satisfaction of deadlines imposed by the implementation. By doing so, some verifications are performed on the composite service before its delivery to the final users. In the other scenario, the composite services are constructed without any consideration of feasible deadlines of the user requirements, namely the implementation is in the form of an untimed automaton. Then, when such composite services have been constructed (through different orchestrators) and offered, the final users can select or apply those, satisfying their possible real-time requirements. This scenario enables the users to test and verify a composite service based on several verifications.

3.4.1 Real-Time Constrains Imposed by Requirements

In this scenario, real-time constraints are already imposed by a target service. Such constraints can be expressed as time intervals to show the deadlines for the execution of operations or can be expressed as delays between two subsequent operations. In the implementation of such scenario in UPPAAL, a local clock is declared in the declaration of the template related to the target service. Then, the clock is appeared as guards on the transitions or as invariants in states of the target service to indicate the deadline for performing an operation or the delay between operations. At each step of an operation execution, if the operation is successfully delegated to an available atomic service through an orchestrator, it means that the time constraints for performing the operation in atomic service satisfies the deadline of the target service. In this end, the formula proposed in Sect. 3.3.2 can be used to check time-actionlocks and the reachability of states in an orchestrator.

Example 3.4.1. Assuming that, the target service depicted in Fig. 3.1 imposes a deadline between 20 and 25 seconds for injecting a low dose insulin and its clock is not reset. Given such a deadline, the following verification was done on the orchestrator o_2 to check whether the operation can be performed by the insulin pump, namely is there any time-actionlock in the orchestrator o_2 from state s_5?

```
E<> not deadlock and orchestrator2.s5
```

The formula was not satisfied. Since the operation is executed by the insulin pump between 30 and 45 seconds. ■

Example 3.4.2. The assumption is that the target service depicted in Fig. 3.1 has no deadlines for the execution of operations and only imposes a delay through which the operation *blood pressure test* should be performed at least 2 minutes after the operation *inject high dose*. Figure 3.7 shows the implementation of the delay in the target service, where t is the clock of the target. It can be seen when the transition `inject_high_dose!` is taken, the clock is reset. Then, when the clock reaches a time greater than or exactly 120 seconds in state t_5, the transition `blood_pressure_test!` can be taken from the state. In order to check whether the closed-loop control system, involving orchestrator o_1, is not subject to a time-actionlock with regards to the delay, the Formula 3.1 was tested and it was satisfied.

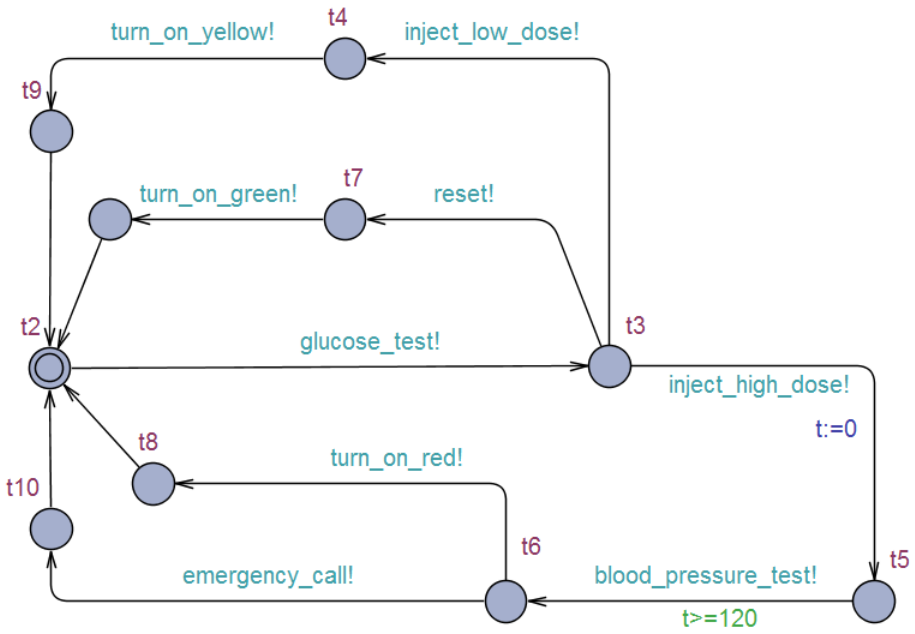


Figure 3.7: Implementing a delay in the target service

Because both states b_1 and b_3 have no invariants and the clock in such states can support an infinity time. ■

3.4.2 Real-Time Expectations Verified by Users

In this scenario, the implementation of a composite service is described by an untimed automaton. This scenario is the case where time constraints are not really critical for the users and based on several composite services that are possibly offered to them. The users are able to test the composite services and finally select those which satisfy their response time expectations for performing some operations. When the states and clocks of available atomic services are accessible, some verification formulas can be performed on the offered composite services by the final users. In UPPAAL, a limited subset of TCTL formulas are available for the aim of verification of such composite services against some properties, including safety, liveness, reachability, and deadlock. Appendix C.2 provides the details about such formulas.

Assuming that, in a composite service, the orchestrator only interacts with one

available atomic service for the realization of the target and the atomic service reset its clock after taking each transition. In such a case, the estimated time for performing a sequence of operations in a path of target service can be tested by the users. In this end, a global clock must be declared in UPPAAL to keep the infimum time, which is passed during the execution of the sequence of operations by the atomic service. This time shows the sum of lower bounds of the time intervals, considered for executing each operation. Notice that, from the state in the target service, where the intended sequence of operations is terminated, the global clock can be reset.

This assumption is, however, restrictive and the idea does not work for the composite service, interacting with more than one atomic service. Assuming that, after the execution of a subset of requested operations by an atomic service, the global clock reaches a time $t \geq \alpha$, where t is the global clock. Furthermore, for the execution of the next requested operation, another atomic service is selected, which imposes a time constraint $x \geq \beta$ for the execution of the operation, where x is the local clock of the atomic service. In such a case, the infimum time of the global clock is updated by the maximum of α and β, instead of updating with sum of them. So, the estimated time of the sequence of operations cannot be tested by the users.

Example 3.4.3. As a verification on the composite service, involving the orchestrator o_1, to check whether the operation of injecting a high dose insulin is performed between 25 and 60 seconds, the following formula was proposed:

```
A[] (target.t5 and Insulin_pump.p5 and orchestrator1.s11)
     imply (Insulin_pump.x≥25 and Insulin_pump.x≤60)
```

This formula was satisfied. Since the insulin pump can inject a high dose of insulin between 30 and 50 seconds. ■

Example 3.4.4. Assuming that, two composite services are offered for the diabetes control system, which one involves the orchestrator o_1 and the other involves the orchestrator o_2 depicted in Fig. 3.2. An IoT user requires a composite service, that necessarily makes an emergency call less than 25 seconds. In this end, the following verifications are done to check which composite service can satisfy the real-time requirement.

```
A[] (target.t10 and Blood_pressure_monitor.b6 and
orchestrator2.s16) imply Blood_pressure_monitor.y<25
```

This formula was not satisfied in the composite service involving o_2, since the orchestrator delegates the emergency call to the blood pressure monitor, and this device perform the operation between 30 and 35 seconds.

```
A[] (target.t10 and Safety_alarm.c5 and orchestrator1.s16)
                    imply Safety_alarm.z<25
```

This formula was satisfied in the composite service involving o_1, since the orchestrator delegates the emergency call to the safety alarm, and this device perform the operation between 10 and 20 seconds. So, this composite service is desirable for the user.

3.5 Contributions

In summary, this chapter makes the following contributions.

- It shows how to exploit a real-time framework to specify time constraints imposed on the operations of both target behavior and available behaviors in the behavior composition framework.
- It proposes a closed-loop control system for the interactions among the main elements of the behavior composition framework, namely the target behavior, orchestrator, and available behaviors.
- It implements the main elements of the behavior composition framework in Uppaal and simulates the interactions among them.
- It describes how to verify the closed-loop control system in Uppaal through several kinds of properties that involve time constraints, and how to detect possible time-actionlocks in a composite service.

Chapter 4

Horizontal Composition with Modular Control

Consider a requirements specification of a dynamic software component in terms of a formal description of its desired behavior and a number of loosely coupled dynamic components with specific behaviors. A composition problem is the realization of a composite component, which conforms to the specification, via a controller. The role of the controller is of great importance in composition of components because it restrains their behavior (control by disablement [51]) or ensures their orchestration (control by delegation [32]), while satisfying the functional requirements specification. Whatever the type of control, modular control naturally arises, since components are assembled in a horizontal and vertical manner as advocated in component-based software development.

This chapter concentrates on a horizontal modular approach in the context of the behavior composition problem. The proposed solution adopts a process-theoretic approach, which seems appropriate when considering the behavior of components and interactions between them. Reaching a horizontal modular control through such a solution consists in four steps summarized as follows.

- Reformulation of the main elements of the behavior composition framework when considering behaviors and orchestrators as process terms.
- Introduction of a new algorithm for automatic construction of orchestrators and new process operators.
- Introduction of three different scenarios for the parallel execution of multiple composite components. The first one focuses on the interleaving of multiple composite components without any interaction. The idea of this scenario came from the multithreading method proposed in [15]. The other scenarios are appropriate for the synchronization of multiple composite components with a unique available component and with distinct available components, respec-

tively. Moreover, deadlock detection is formally defined for the last scenarios.

– Extension of horizontal composition with modular control. Besides, a theoretical issue is investigated when utility values are assigned to orchestrators.

4.1 The Process-Theoretic Framework

The elements involved in the behavior composition problem can be represented by (closed or ground) process terms of some languages described by signatures. They exhibit behaviors formally deduced from the structural operational semantics of the operators of languages. The following description is mainly based on the mathematical formalism borrowed from [6], but without referring to any specific process theory (e.g., CCS, CSP, ACP) to simplify the presentation.

Definition 4.1.1. (page 11 in [6]) A signature Σ is a set of constant symbols and function symbols with their arities (operators with a finite number of operands).

Definition 4.1.2. (page 12 in [6]) The set of all terms over a signature Σ and a set of variables V, denoted by $\mathcal{T}(\Sigma, V)$, is the smallest set that satisfies the following conditions:

– each variable in V is a term in $\mathcal{T}(\Sigma, V)$;

– each constant in Σ is a term in $\mathcal{T}(\Sigma, V)$;

– in the case that f is an n-ary operator ($n \geq 1$) and $t_1, \ldots, t_n$ are terms in $\mathcal{T}(\Sigma, V)$, then $f(t_1, \ldots, t_n)$ is a term in $\mathcal{T}(\Sigma, V)$.

For simplification, the set V is discarded from the notation and the set $\mathcal{T}(\Sigma, V)$ is denoted by $\mathcal{T}(\Sigma)$. A term without variables is called a closed (or ground) term, and the set of all closed terms is denoted by $\mathcal{C}(\Sigma)$.

For the rest of this chapter, A denotes a given finite set of actions (operations) and A^* is the set of words over A. Those involved in a process term p is denoted by $\alpha(p)$, which is the alphabet of p.

Definition 4.1.3. (page 12 in [6]) Let Σ and V be a signature and a set of variables, respectively. A substitution θ is a mapping from V to $\mathcal{T}(\Sigma, V)$. For any term $t \in \mathcal{T}(\Sigma, V)$, $t[\theta]$ denotes the term obtained by the simultaneous substitution of all variables in t with respect to θ, that is,

– for each variable $x \in V$, $x[\theta] = \theta(x)$,

– for each constant $c \in \Sigma$, $c[\theta] = c$, and

– for any n-ary function symbol f $(n \geq 1)$ and terms $t_1, \ldots, t_n$ in $\mathcal{T}(\Sigma, V)$, $f(t_1, \ldots, t_n)[\theta]$ is the term $f(t_1[\theta], \ldots, t_n[\theta])$.

In this chapter, the intended signatures include two constant symbols 0 and 1, which represent the process that does nothing (it can be only deadlock) and the process that terminates successfully, respectively. Moreover, they have a unary operator $a.p$ representing the process that executes action $a \in A$ and proceeds as p. According to a signature, the operators are chosen among "$\cdot$", "$+$", "$\oplus$", "$\Box$", "*", "$||$", "$|$", and "$|[\_]|$", which denote the concatenation, nondeterministic choice, deterministic choice, external choice, iteration, interleaving (parallel composition), synchronous parallel composition, and interface parallel composition, respectively.

4.1.1 Labelled Transition System

Definition 4.1.4. (page 35 in [6]) A transition-system space over a set of actions A is a set of states S, equipped with one ternary relation $\rightarrow$ and one subset $\downarrow$:

– $\rightarrow \subseteq S \times A \times S$ is the subset of transitions;

– $\downarrow \subseteq S$ is the set of final states.

The notation $s \xrightarrow{a} t$ is used for $\langle s, a, t \rangle \in \rightarrow$.

Definition 4.1.5. (page 36 in [6]) The reachability relation $\rightarrow^* \subseteq S \times A^* \times S$ is inductively defined as follows:

– $r \xrightarrow{\epsilon}{}^* r$ for each $r \in S$, where $\epsilon \in A^*$ denotes the empty word;

– for all $r, s, t \in S$, $w \in A^*$, and $a \in A$, if $r \xrightarrow{w}{}^* s$ and $s \xrightarrow{a} t$, then $r \xrightarrow{wa}{}^* t$.

A state $s \in S$ is said to be reachable from $r \in S$ if and only if there is a word $w \in A^*$ such that $r \xrightarrow{w}{}^* s$.

Definition 4.1.6. (page 37 in [6]) For each state $s \in S$, the transition system induced by s contains all states reachable from s, and it has the transitions and final states induced by the transition-system space. State s is then the initial state of the transition system.

The next two definitions provide a specialization of the relation $\rightarrow$ and subset $\downarrow$ introduced in Def. 4.1.4 for the case where $S = \mathcal{T}(\Sigma)$.

Definition 4.1.7. [7] Given a set of terms $\mathcal{T}(\Sigma)$ and a set of actions A, a transition relation between two terms on a given action is denoted by "$\rightarrow$", where $\rightarrow \subseteq \mathcal{T}(\Sigma) \times A \times \mathcal{T}(\Sigma)$. Given a process $p \in \mathcal{T}(\Sigma)$, $\langle p, a, p' \rangle \in \rightarrow$ is denoted by $p \xrightarrow{a} p'$. The predicates $p \xrightarrow{a}$ and $p \not\xrightarrow{a}$ are used to indicate that p has or does not have a transition labeled by $a \in A$, respectively.

Definition 4.1.8. [7] Given a set of terms $\mathcal{T}(\Sigma)$, a successful termination predication over terms is denoted by "$\downarrow$" such that $\downarrow \subseteq \mathcal{T}(\Sigma)$. For a process $p \in \mathcal{T}(\Sigma)$, $p \in \downarrow$ is written $p\downarrow$.

Definition 4.1.9. (page 51 in [6]) A deduction system is a pair (Σ,R), where Σ is a signature and R is a set of rules of the form of

$$\frac{\Phi}{\psi}.$$

The premises of the rule Φ is a set of formulas and the conclusion of the rule ψ is a formula. A formula is either a "transition pattern" $p \xrightarrow{a} p'$ or a "termination pattern" $p\downarrow$, where $p, p' \in \mathcal{T}(\Sigma)$ and $a \in A$.

Definition 4.1.10. (page 51 in [6]) Let (Σ,R) be a deduction system. The transition-system space induced by (Σ,R) is the quadruple $(\mathcal{C}(\Sigma), A, \rightarrow, \downarrow)$, where $\rightarrow \subseteq \mathcal{C}(\Sigma) \times A \times \mathcal{C}(\Sigma)$ or $\downarrow \subseteq \mathcal{C}(\Sigma)$ contains a formula if and only if there is a rule $\frac{\Phi}{\psi} \in$ R and a substitution θ such that the formula is a closed substitution instance of ψ (i.e., $\psi[\theta]$) when $\phi[\theta] \in \rightarrow$ or $\phi[\theta] \in \downarrow$ for all $\phi \in \Phi$.

Definition 4.1.11. Let (Σ,R) and $p \in \mathcal{C}(\Sigma)$ be a deduction system and a closed process term, respectively. The transition system induced by p with respect to (Σ,R) is a 5-tuple $\mathcal{TS}_p = (\mathcal{C}(p, \Sigma), \alpha(p), \rightarrow, p, \downarrow)$, where $\mathcal{C}(p, \Sigma) \subseteq \mathcal{C}(\Sigma)$ is the set of states reachable from p, $\alpha(p) \subseteq A$ is the set of actions in p, $\rightarrow \subseteq \mathcal{C}(p, \Sigma) \times \alpha(p) \times \mathcal{C}(p, \Sigma)$ is the transition relation, p is the initial state, and $\downarrow$ is the set of final states.

4.1.2 Operational Semantics Rules

Based on the predicates defined in Def. 4.1.7 and 4.1.8, the operational semantics rules of the constant and operator symbols are given as follows, where $a \in A$.

$$\frac{}{\neg 0\downarrow \wedge 0 \not\xrightarrow{a}}, \tag{R1}$$

$$\frac{}{1\downarrow}, \tag{R2}$$

$$\frac{}{a.p \xrightarrow{a} p}, \tag{R3}$$

$$\frac{p \xrightarrow{a} p'}{p{\cdot}q \xrightarrow{a} p'{\cdot}q}, \quad \frac{p\downarrow, q \xrightarrow{a} q'}{p{\cdot}q \xrightarrow{a} q'}, \tag{R4}$$

$$\frac{p\downarrow}{(p+q)\downarrow}, \quad \frac{q\downarrow}{(p+q)\downarrow}, \quad \frac{p \xrightarrow{a} p'}{p+q \xrightarrow{a} p'}, \quad \frac{q \xrightarrow{a} q'}{p+q \xrightarrow{a} q'}, \tag{R5}$$

$$\frac{}{p^*\downarrow}, \quad \frac{p \xrightarrow{a} p'}{p^* \xrightarrow{a} p'{\cdot}p^*}, \tag{R6}$$

$$\frac{p \xrightarrow{a} p'}{p||q \xrightarrow{a} p'||q}, \quad \frac{q \xrightarrow{a} q'}{p||q \xrightarrow{a} p||q'}, \tag{R7}$$

$$\frac{p \xrightarrow{a} p', q \xrightarrow{a} q'}{p|q \xrightarrow{a} p'|q'}, \text{ with } \alpha(p) = \alpha(q), \tag{R8}$$

$$\frac{p \xrightarrow{a} p', a \notin H}{p|[H]|q \xrightarrow{a} p'|[H]|q}, \quad \frac{q \xrightarrow{a} q', a \notin H}{p|[H]|q \xrightarrow{a} p|[H]|q'}, \quad \frac{p \xrightarrow{a} p', q \xrightarrow{a} q', a \in H}{p|[H]|q \xrightarrow{a} p'|[H]|q'}, \ H \subseteq A, \tag{R9}$$

$$\frac{p\downarrow, q\downarrow}{(p \odot q)\downarrow}, \text{ for } \odot = \text{“}\cdot\text{”}, \text{“}||\text{”}, \text{“}|\text{”}. \tag{R10}$$

In Rules R9, the operator “$|[\_]|$” is like the synchronous parallel composition, except that synchronization occurs on actions that belong to a common action set $H \subseteq A$.

Choice Operators

Rules R5 provides the operational semantic rules for the nondeterministic choice operator. However, two other operators, namely "$\oplus$" and "$\Box$", are also necessary to provide a better process-based view to behavior composition. More precisely, since the target behavior in the behavior composition framework is deterministic by assumption, the operator "+" cannot be used. In this end, the operator "$\oplus$" is used instead. It has the same semantic rules, but with a restriction in its premises. That is, if the predicate $p \xrightarrow{a}$ (resp. $q \xrightarrow{a}$) holds then $q \not\xrightarrow{a}$ (resp. $p \not\xrightarrow{a}$) holds. Otherwise, the process $p \oplus q$ is deadlocked.

Contrary to the operator "+" in which the choice is an internal choice, the operator "$\Box$" is used in the case where the choice is an external choice. Its semantic rules are totally the same as those in Rules R5. [1]

In comparison with the process algebra used in [7], the proposed calculus is more complete. In [7], the operator "+" is the only choice operator and it is a nondeterministic internal choice. Besides, a deterministic version, namely $det(p)$, was defined for a nondeterministic process p. Furthermore, the concatenation and iteration operators were eschewed. These restrictions are too severe when considering the behavior composition framework, since in that case, behaviors are restricted to be only internal choices of sequences of operations.

Example 4.1.1. Given the services of online news providers already presented in Example 2.1.1, the modeling of behaviors by process terms is illustrated. The following process terms describe behaviors of four available services s_1 to s_4 and two composite components (requirements) j and r: [2]

1. In [6], some semantics rules were provided for the operator "$\Box$" with respect to a silent action "τ". However, this action is not explicitly used hereafter in the description of behaviors.

2. For the simplification of illustration, the constant symbol 1 was discarded in some process terms describing behaviors. For instance, the process term s_3 with respect to the syntactical rules is: s_3: $(archive.1 + publish.1 + upload\_video.(1 + archive.1))^*$.

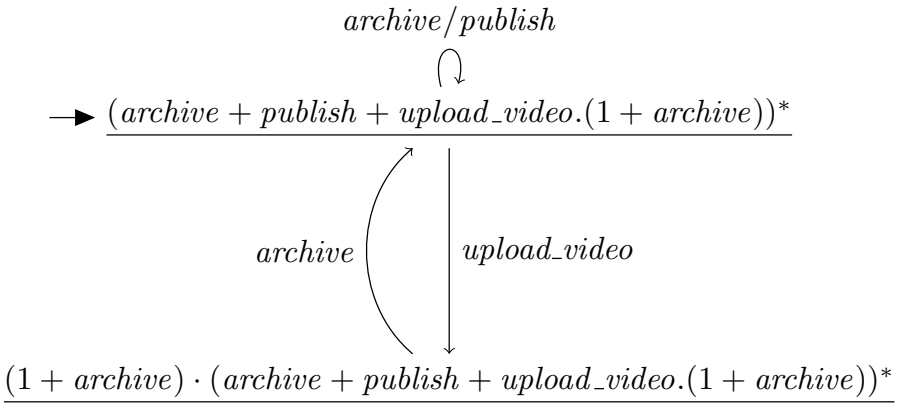


Figure 4.1: The transition system induced by s_3

s_1: $(upload\_video.(1 + (put\_caption + archive)\cdot(1 + archive)))^*$,

s_2: $(write\_story.(translate + (translate.(1 + archive))))^*$,

s_3: $(archive + publish + upload\_video.(1 + archive))^*$,

s_4: $(upload\_photo.put\_caption.translate)^*$,

j: $(write\_story.translate.archive.publish)^*$,

r: $((upload\_video \oplus (upload\_photo.put\_caption.translate))\cdot archive)^*$.

The behaviors of available services s_1 to s_4 were already represented by the available behaviors $\mathcal{B}_1$ to $\mathcal{B}_4$ in Example 2.1.1, and the composite components j and r by the target behaviors $\mathcal{B}_{t1}$ and $\mathcal{B}_{t2}$, respectively.

Figure 4.1 shows the transition graph of the transition systems induced by s_3.[3] The alphabet of s_4 is $\alpha(s_4) = \{put\_caption, translate, upload\_photo\}$.

■

3. A transition labeled with actions separated by a slash is just a more compact notation to represent multiple transitions. The initial state is indicated by a dark arrow. The final states are underline.

4.1.3 Trace, Simulation, and Bisimulation Equivalences

Since this chapter does not refer to any process theory, the syntactical equalities of terms is not used. In fact, it subsumes the notion of derivation, where an axiom of the theory or a proof rule is applied at each step of a derivation. Trace, simulation, and bisimulation equivalences are used instead.

Definition 4.1.12. (page 394 in [6]) Let $p \in \mathcal{C}(\Sigma)$ be a closed process term and $\mathcal{TS}_p$ be the transition system induced by p. A sequence (word) $w \in A^*$ is a trace of $\mathcal{TS}_p$ if and only if there is a state $r \in \mathcal{C}(p, \Sigma)$ with $p \xrightarrow{w}{}^* r$. The sequence w is an accepting trace of $\mathcal{TS}_p$ if $r\downarrow$.

The trace set of $\mathcal{TS}_p$, denoted $\mathbb{T}(\mathcal{TS}_p)$, is the non-empty prefix-closed set of traces of $\mathcal{TS}_p$.

Definition 4.1.13. Let $\mathbb{T}(\mathcal{TS}_p)$ and $\mathbb{T}(\mathcal{TS}_q)$ be the trace sets of $\mathcal{TS}_p$ and $\mathcal{TS}_q$, respectively. The processes p and q are trace equivalent, denoted $p \simeq_{\mathrm{T}} q$, if and only if $\mathcal{TS}_p$ and $\mathcal{TS}_q$ have the same trace sets (i.e., $\mathbb{T}(\mathcal{TS}_p) = \mathbb{T}(\mathcal{TS}_q)$).

Definition 4.1.14. Let $p, q \in \mathcal{C}(\Sigma)$ be two closed process terms. A simulation for $\langle \mathcal{TS}_p, \mathcal{TS}_q \rangle$ is a binary relation $R \subseteq \mathcal{C}(p, \Sigma) \times \mathcal{C}(q, \Sigma)$, such that:

- $\langle p, q \rangle \in R$;
- for all $\langle r, s \rangle \in R$ the following conditions hold:
  - if $r\downarrow$, then $s\downarrow$,
  - for all $a \in \alpha(p)$ and $r' \in \mathcal{C}(p, \Sigma)$ such that $r \xrightarrow{a} r'$, $s \xrightarrow{a} s'$ for some $s' \in \mathcal{C}(q, \Sigma)$ with $\langle r', s' \rangle \in R$.

Definition 4.1.15. The process p is simulated by the process q, denoted $p \preceq_R q$, if there exists a simulation for $\langle \mathcal{TS}_p, \mathcal{TS}_q \rangle$.

Remark 4.1.1. (transitivity) Let $p, q, r \in \mathcal{C}(\Sigma)$ be process terms such that $p \preceq_{R_1} q$ and $q \preceq_{R_2} r$. The process p is simulated by the process r, $p \preceq_R r$, where $R = R_1 \circ R_2$.

Definition 4.1.16. Let $p, q \in \mathcal{C}(\Sigma)$ be two closed process terms. A bisimulation for $\langle \mathcal{TS}_p, \mathcal{TS}_q \rangle$ is a binary relation $R \subseteq \mathcal{C}(p, \Sigma) \times \mathcal{C}(q, \Sigma)$ such that:

- $\langle p, q \rangle \in R$;

– for all $\langle r, s\rangle \in R$ the following conditions hold:
  – $r\!\downarrow$ if and only if $s\!\downarrow$,
  – for all $a \in \alpha(p)$ and $r' \in \mathcal{C}(p, \Sigma)$ such that $r \xrightarrow{a} r'$, $s \xrightarrow{a} s'$ for some $s' \in \mathcal{C}(q, \Sigma)$ with $\langle r', s'\rangle \in R$,
  – for all $a \in \alpha(q)$ and $s' \in \mathcal{C}(q, \Sigma)$ such that $s \xrightarrow{a} s'$, $r \xrightarrow{a} r'$ for some $r' \in \mathcal{C}(p, \Sigma)$ with $\langle r', s'\rangle \in R$.

Definition 4.1.17. The processes p and q are bisimilar, denoted $p \leftrightarrows_R q$, if there exists a bisimulation for $\langle \mathcal{TS}_p, \mathcal{TS}_q\rangle$.

Remark 4.1.2. (transitivity) Let $p, q, r \in \mathcal{C}(\Sigma)$ be process terms such that $p \leftrightarrows_{R_1} q$ and $q \leftrightarrows_{R_2} r$. The processes p and r are bisimilar, $p \leftrightarrows_R r$, where $R = R_1 \circ R_2$.

Remark 4.1.3. (transitivity) Let $p, q, r \in \mathcal{C}(\Sigma)$ be process terms such that $p \leftrightarrows_{R_1} q$ (resp. $p \preceq_{R_1} q$) and $q \preceq_{R_2} r$ (resp. $q \leftrightarrows_{R_2} r$) , where R_1 and R_2 are a bisimulation (resp. simulation) relation and a simulation (resp. bisimulation) relation, respectively. The process p is simulated by the process r, $p \preceq_R r$, where $R = R_1 \circ R_2$.

4.2 A Process-Based View to Behavior Composition

The specification of nondeterministic behaviors with process terms is done from the following signature: [4]

$$\Sigma_s := \{1,\ a._-,\ \_\cdot\_,\ \_+\_,\ \_^*\}.$$

Even if the behaviors of available components s_1 to s_n are separate entities, they must be considered as a whole in order to be able to synthesize an orchestrator. The system behavior is obtained by carrying out a parallel composition with no interaction between behaviors: $s = s_1 \,|||\, \cdots \,|||\, s_n = |||_{i=1}^{n} s_i$. The operational semantics of the operator "$|||$" is given by the following rules:

$$\frac{p_1\!\downarrow, \ldots, p_n\!\downarrow}{p\!\downarrow}, \quad \frac{p_k \xrightarrow{a} p'_k}{p \xrightarrow{\langle a,k\rangle} p'}, \text{ for all } k \in I_n, \tag{R11}$$

4. The constant symbol 0 is ignored in the signatures used to specify the system, target, and orchestrator, as explicit deadlock is forbidden in the specification of target and available behaviors.

where $I_n = \{1, \ldots, n\}$, $p = |||_{i=1}^{n} p_i$, and $p' = |||_{i=1}^{n} p'_i$ such that $p'_i = p_i$, $i \neq k$. The semantic rules of this operator is different with "||", since the transition's label in the conclusion of "|||" contains a pair involving an operation and the index of an available atomic component executing it. When applied to s, the first rule of R11 asserts that the system terminates as soon as all the components terminate immediately and the second rule of R11 indicates that a component evolves independently of the other components in the system. It should be noted that, the system's alphabet is an extension of the alphabets of all the components: $\alpha(s) = \left(\cup_i \ \alpha(s_i) \right) \times I_n$. The reason for this choice is twofold. First, components can share some operations on which they must not synchronize. Therefore, the parallel composition is well defined and does not introduce nondeterminism, since $\langle a, i \rangle \neq \langle a, j \rangle$ if $i \neq j$. Second, the indices included in actions that belong to $\alpha(s)$ represent useful information used in the construction of orchestrators based, among other things, on the behavior of the overall system. More precisely, they suggest possible choices between the appropriate component operations offered by different providers.

Traces of actions that belong to $\alpha(s)^*$ can be projected onto traces of actions that belong to $(\cup_i \alpha(s_i))^*$ by a projector operator π_1.

Definition 4.2.1. Let $w \in (A \times I_n)^*$. A projector operator π_1 recursively defined as follows:

(i) $\pi_1(\epsilon) := \epsilon$;

(ii) $\pi_1(\langle a, k \rangle w) := a\pi_1(w)$.

The function π_1 is extended to an arbitrary set of traces in the usual way: $\pi_1(W) := \{\pi_1(w) \mid w \in W\}$, where $W \subseteq (A \times I_n)^*$. By convention, $\pi_1(w) = w$, when actions of w are singletons instead of pairs (i.e., $w \in A^*$).

Based on Def. 4.2.1, trace equivalence can be refined to *index-less trace equivalence*.

Definition 4.2.2. The processes p and q are index-less trace equivalent, denoted $p \simeq_{\mathrm{T}}^{I_n} q$, if and only if $\pi_1(\mathbb{T}(\mathcal{TS}_p)) = \pi_1(\mathbb{T}(\mathcal{TS}_q))$.

The largest ND-simulation relation, defined in Sect. 2.2 (see Def. 2.2.1), is reformulated in the context of the process-theoretic approach.

Definition 4.2.3. Let $p, q \in \mathcal{C}(\Sigma)$ be two closed process terms. An ND-simulation relation for $\langle \mathcal{TS}_p, \mathcal{TS}_q \rangle$ is a binary relation $R \subseteq \mathcal{TS}_p \times \mathcal{TS}_q$, if for all $r \in \mathcal{C}(p, \Sigma)$ and $s \in \mathcal{C}(q, \Sigma)$ such that $(r, s) \in R$ the following conditions hold:

if $r\downarrow$, then $s\downarrow$; (1)

if $r \xrightarrow{a} r'$ for some $a \in \alpha(p)$, then

there exists $s' \in \mathcal{C}(q, \Sigma)$ and $a' \in \alpha(q)$ with $\pi_1(a) = \pi_1(a')$ such that $s \xrightarrow{a'} s'$ (2)

and for all $s' \in \mathcal{C}(q, \Sigma)$ such that $s \xrightarrow{a'} s'$, (3)

$(r', s') \in R$.

Definition 4.2.4. The process p is ND-simulated by the process q, denoted $p \preceq_R^{\mathrm{ND}} q$, if there exists an ND-simulation for $\langle \mathcal{TS}_p, \mathcal{TS}_q \rangle$.

Definition 4.2.5. A binary relation R is an *index-less simulation* relation, if condition (3) is removed from Def. 4.2.3. A process p is index-less simulated by q, denoted $p \preceq_R^{I_n} q$, if there exists an *index-less simulation* for $\langle \mathcal{TS}_p, \mathcal{TS}_q \rangle$.

An ND-simulation relation imposes stronger constraints because the process q must simulate the process p whatever its behavior due to nondeterminism on a same action. An index-less simulation relation R is established in the same way as a simulation relation, except that a match between transitions occurs when their actions are the same after projection.

Definition 4.2.6. Two processes p and q are *index-less bisimilar*, denoted $p \leftrightarrows_R^{I_n} q$, if and only if there exists an index-less simulation relation R such that $p \preceq_R^{I_n} q$ and $q \preceq_R^{I_n} p$.

Remark 4.2.1. Let $p, q, r \in \mathcal{C}(\Sigma)$ be process terms such that $\alpha(p) = \alpha(r)$, $p \leftrightarrows_{R_1}^{I_n} q$ and $q \leftrightarrows_{R_2}^{I_n} r$, where R_1 and R_2 are index-less bisimulation relations. The processes p and r are bisimilar, $p \leftrightarrows_R r$, where $R = R_1 \circ R_2$.

Remark 4.2.2. Let $p, q, r \in \mathcal{C}(\Sigma)$ be process terms such that $p \leftrightarrows_{R_1}^{I_n} q$ and $q \preceq_{R_2} r$, where R_1 and R_2 are an index-less bisimulation relation and a simulation relation, respectively. The process p is index-less simulated by the process r, $p \preceq_R^{I_n} r$, where $R = R_1 \circ R_2$.

Remark 4.2.3. Let $p, q, r \in \mathcal{C}(\Sigma)$ be process terms such that $\alpha(p) = \alpha(r)$, $p \leftrightarrows^{I_n}_{R_1} q$ and $q \preceq^{I_n}_{R_2} r$, where R_1 and R_2 are an index-less bisimulation relation and an index-less simulation relation, respectively. The process p is simulated by the process r, $p \preceq_R r$, where $R = R_1 \circ R_2$.

Remark 4.2.4. Let $p, q, r \in \mathcal{C}(\Sigma)$ be process terms such that $p \leftrightarrows^{I_n}_{R_1} q$ and $q \leftrightarrows_{R_2} r$, where R_1 and R_2 are an index-less bisimulation relation and a bisimulation relation, respectively, The processes p and r are index-less bisimilar, $p \leftrightarrows^{I_n}_{R} r$, where $R = R_1 \circ R_2$.

The results of these remarks are used in the proofs of some lemmas stated in Sect. 4.2.2.

4.2.1 Synthesis of Orchestrators

Let t be the process term representing the expected behavior (implementation) of a component. It is defined from another signature $\Sigma_t := \{1,\ a._-,\ \_\cdot\_,\ \_\oplus\_,\ \_^*\}$, which restricts the implementation to have deterministic behavior. This is not the case for the behavior (interface) of the available components. [5] Furthermore, $\alpha(t) \subseteq \pi_1(\alpha(s))$.

According to Def. 4.2.3, [6] the set of ND-simulation relations of t by s forms a join-semilattice. The least upper bound is the largest ND-simulation relation.

Definition 4.2.7. An ND-simulation relation $R^{\min}$ of t by s, denoted $t \preceq^{\mathrm{ND}}_{R^{\min}} s$, is said to be *minimal* if there exists no other ND-simulation relation R' of t by s such that $R' \subseteq R^{\min}$.

Remark 4.2.5. Let $p, q, r \in \mathcal{C}(\Sigma)$ be three process terms such that $p \leftrightarrows^{I_n}_{R_1} q$ and $q \preceq^{\mathrm{ND}}_{R_2^{\min}} r$, where R_1 and $R_2^{\min}$ are an index-less bisimulation relation and a minimal ND-simulation relation, respectively. The process p is simulated by r, $p \preceq_R r$, where $R = R_1 \circ R_2^{\min}$.

5. Typically, the implementation of a component has a deterministic behavior. Its interface is an abstraction of its implementation, which hides (erases) actions of the implementation. So, an interface generally has a nondeterministic behavior.

6. To be more precise, in Def. 4.2.3, $p \in \mathcal{C}(\Sigma_t)$ and $q \in \mathcal{C}(\Sigma_s \cup \{|||\})$, that is $\Sigma = \Sigma_t \cup \Sigma_s \cup \{|||\}$.

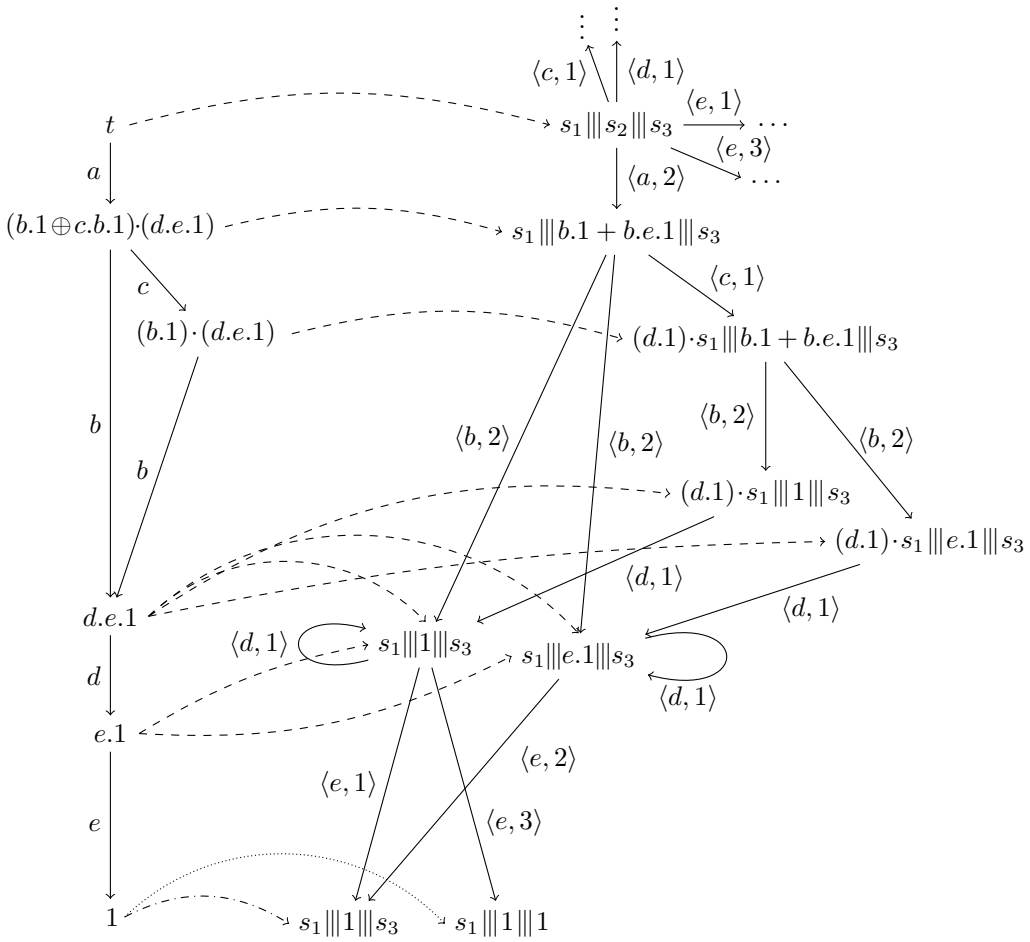


Figure 4.2: The largest ND-simulation relation of t by s

Example 4.2.1. Let $t = a.(b.1 \oplus c.b.1)\cdot(d.e.1)$ be the composite component to be realized from the following available components:

$$\begin{aligned} s_1 &= (d.1 + e.1 + c.d.1)^*, \\ s_2 &= a.(b.1 + b.e.1) + 1, \\ s_3 &= e.1 + 1. \end{aligned}$$

It should be noted the difference between $a.1 + 1$ and a^*. Contrary to the latter, the former can execute a once at most. Therefore, the component s_3 can terminate promptly or be invoked only once to perform e and terminates. Furthermore, it is

important to specify that each component can terminate promptly, particularly if it is not used by the composite component, due to condition (1) in Def. 4.2.3.

Figure 4.2 shows the transition system induced by t and a part of the one induced by s. The dashed, dotted, and dash-dotted arrows reveal the largest ND-simulation relation R of t by s. There are two minimal ND-simulation relations: $R_1^{\min}$ (given by dashed and dash-dotted arrows) and $R_2^{\min}$ (given by dashed and dotted arrows). More precisely,

$$R_1^{\min} = R \setminus \{(1, s_1 \| 1 \| 1)\} \text{ and } R_2^{\min} = R \setminus \{(1, s_1 \| 1 \| s_3)\}.$$

■

Theorem 4.2.1. Let s and t be a system and a composite component, respectively. If $t \preceq^{\mathrm{ND}}_{R^{\min}} s$, then there exists an orchestrator that realizes t on s.

Proof. The proof follows essentially the same lines as the proof of the analogous theorem 1 of [32] with arguments on transition systems (i.e., $\mathcal{TS}_t$ and $\mathcal{TS}_s$ in our case) to exhibit the control policies of the orchestrator. □

Remark 4.2.6. For a given composite component t and a given system s, there are as many orchestrators as there are minimal ND-simulation relations of t by s, say $R_1^{\min}, \ldots, R_m^{\min}$. In the original behavior composition framework [32], the controller generator is nondeterministic and is not necessarily equal to $o_1 \,\square\, \cdots \,\square\, o_m$, even if the largest ND-simulation relation is equal to $R_1^{\min} \cup \cdots \cup R_m^{\min}$, where $o_1, \ldots, o_m$ are, respectively, the orchestrators inferred from $R_1^{\min}, \ldots, R_m^{\min}$. The process operator "$\square$" picks an orchestrator to realize t.

Algorithm 2 includes a lifting procedure to obtain a process term o for the orchestrator from a minimal ND-simulation relation $R^{\min}$ of t by s. It consists in translating the terms over the signature Σ_t with alphabet $\alpha(t)$ into terms over the signature $\Sigma_o = \{1,\ \vee_i\, [c_i{:}\langle a, k_i\rangle].\_,\ \_\cdot\_,\ \_\oplus\_,\ \_^*\}$, with alphabet included in $\alpha(s)$. The semantics of the guarded version of "$\langle a, k\rangle$.", called the *control* operator, is given by the

Algorithm 2 Lifting procedure for the construction of an orchestrator from a minimal ND-simulation relation $R^{\min}$ of t by s.

global: $\mathcal{TS}_t, \mathcal{TS}_s$
initial call: $\mathit{Lift}(R^{\min}, t \in \mathcal{C}(\Sigma_t), t \in \mathcal{C}(t, \Sigma_t))$
output: $o \in \mathcal{C}(\Sigma_o)$
procedure $\mathit{Lift}(R, p, r)$
  switch p **do**
    case $p = 1$
      return 1
    case $p = a.p'$
      return $\vee_i [q_i{:}\langle a, k_i\rangle].\mathit{Lift}(R \setminus Q \cup \{\langle r', q_i'\rangle \mid q_i \xrightarrow{\langle a,k_i\rangle} q_i' \in \mathcal{TS}_s\}, p', r')$,
        with $\langle r, q_i\rangle \in R$, where $r \xrightarrow{a} r' \in \mathcal{TS}_t$ and
        $Q = \{\langle r', q'\rangle \mid r \xrightarrow{a} r' \in \mathcal{TS}_t$ with $\langle r', q'\rangle \in R\}$
    case $p = p_1 \oplus p_2$
      return $\mathit{Lift}(R, p_1, r) \oplus \mathit{Lift}(R, p_2, r)$
    case $p = p_1 \cdot p_2$
      return $\mathit{Lift}(R, p_1, r) \cdot \mathit{Lift}(R, p_2, p_2)$
    case $p = p_1^*$
      return $\mathit{Lift}(R, p_1, r)^*$
end procedure

following rule: [7]

$$\frac{c_j}{(\vee_i [c_i{:}\langle a, k_i\rangle].p_i) \xrightarrow{\langle a,k_j\rangle} p_j}, \tag{R13}$$

where i varies from 1 to l, $1 \leq j \leq l$, and $c_1, \ldots, c_l$ are conditions on the system's current state, more precisely process terms that belong to $\mathcal{TS}_s$. The lifting procedure is implemented as a recursive function, where its parameters are as follows.

- The first parameter is a minimal ND-simulation relation.
- The second parameter is a process term that belongs to $\mathcal{C}(\Sigma_t)$.
- The third parameter is a state that belongs to $\mathcal{C}(t, \Sigma_t)$.

7. Contrary to the nondeterministic orchestrators in the original behavior composition framework, the control operator leads, as if by magic, to deterministic orchestrators. Hence, the operator "+" has been replaced by "$\oplus$" in Σ_o, in conjunction with the addition of the control operator.

For each process operator belonging to Σ_t, a corresponding case is provided in Algorithm 2.

In the case $p = a.p'$, r and r' are states that correspond to process term p and subterm p', respectively. Let $Q' = \{q' \in \mathcal{C}(s, \Sigma_s \cup \{|||\}) \mid \langle r', q' \rangle \in R\}$ be the set of states that simulate r'. For all i, $1 \leq i \leq l$, let $q_i \in \mathcal{C}(s, \Sigma_s \cup \{|||\})$ be a state such that $\langle r, q_i \rangle \in R$ (i.e., l states simulate r). After the execution of operation $\langle a, k_i \rangle$ by the system after the request of a by the implementation, there can be some states included in Q' that are not directly reachable from q_i on a ($1 \leq i \leq l$). Hence, in the recursive call, the relation R', where $R' = R \setminus Q \cup \{\langle r', q_i' \rangle \mid q_i \xrightarrow{\langle a, k_i \rangle} q_i' \in \mathcal{TS}_s\}$, must only maintain the pairs $\langle r', q_i' \rangle \in R$, where q_i' is directly reachable from q_i. This is the reason to provide a new relation as the first argument for the recursive call. For instance, in Fig. 4.2, the corresponding state of the subterm $b.1$ is $(b.1) \cdot (d.e.1)$. This state is simulated by only one state, that is, $(d.1) \cdot s_1 ||| b.1 + b.e.1 ||| s_3$. When the operation b is executed, the next state is $(d.e.1)$. Four states in the system, namely $(s_1 ||| 1 ||| s_3)$, $(s_1 ||| e.1 ||| s_3)$, $((d.1) \cdot s_1 ||| 1 ||| s_3)$, and $((d.1) \cdot s_1 ||| e.1 ||| s_3)$ simulate this state. However, both pairs $\langle d.e.1, s_1 ||| 1 ||| s_3 \rangle$ and $\langle d.e.1, s_1 ||| e.1 ||| s_3 \rangle$ must be removed from R for the recursive call.

In the case $p = p_1 . p_2$, the first and last parameters in the lift procedure call located in the left side of the concatenation operator, are the same as the corresponding input parameters. However, the last parameter in the lift procedure call located in the right side of the concatenation operator is different, since the corresponding state of subterm p_2 is p_2, not r.

For the other cases, the first and last parameters in the lift procedure calls are the same as the corresponding input parameters, because the subterms coincide with the input state r.

The orchestrator $o = \mathit{Lift}(R^{\min}, t, t)$ is then inferred from $R^{\min}$. The term $\vee_i [q_i : \langle a, k_i \rangle].\mathit{Lift}(R'^{\min}, p', r')$ clearly reveals that the orchestrator's decision is based on the observation of the system's current state. If the latter is q_j, the orchestrator delegates the operation a to the component s_{k_j}. This is a *state-feedback* control by delegation. It differs from the one adopted in [32], which is based on histories. Conditions that appear in the control operator are mutually exclusive. By construction, an orchestrator can only authorize a single component to execute an operation requested by

the composite component at each step of its evolution. It has no choice in the way it realizes the composite component t. In some sense, the orchestrator is atomic. For conciseness reasons, simplifications are done on guards. The condition is removed when there is only one possibility for delegation because the orchestrator implicitly knows the system's current state. Essentially, an orchestrator gives a dynamic interpretation to $R^{\min}$ and allows for discarding it in the next sections.

Example 4.2.2. The orchestrator inferred by the minimal ND-simulation relation $R_1^{\min}$ of t by s established in Example 4.2.1 is calculated as follows:

$$
\begin{aligned}
o_1 &= \mathit{Lift}(R_1^{\min}, a.(b.1 \oplus c.b.1)\cdot(d.e.1), a.(b.1 \oplus c.b.1)\cdot(d.e.1)) \\
&= \langle a,2\rangle.\mathit{Lift}(R_1^{\min}, (b.1 \oplus c.b.1)\cdot(d.e.1), (b.1 \oplus c.b.1)\cdot(d.e.1)) \\
&= \langle a,2\rangle.(\mathit{Lift}(R_1^{\min}, b.1 \oplus c.b.1, (b.1 \oplus c.b.1)\cdot(d.e.1)) \cdot \mathit{Lift}(R_1^{\min}, d.e.1, d.e.1)) \\
&= \langle a,2\rangle.(\mathit{Lift}(R_1^{\min}, b.1, (b.1 \oplus c.b.1)\cdot(d.e.1)) \oplus \mathit{Lift}(R_1^{\min}, c.b.1, (b.1 \oplus c.b.1)\cdot(d.e.1)))\cdot \\
&\quad \mathit{Lift}(R_1^{\min}, d.e.1, d.e.1) \\
&= \langle a,2\rangle.(\langle b,2\rangle.\mathit{Lift}(R_1^{\min}, 1, d.e.1) \oplus \langle c,1\rangle.\mathit{Lift}(R_1^{\min}, b.1, (b.1)\cdot(d.e.1))) \cdot (\langle d,1\rangle. \\
&\quad \mathit{Lift}(R_1^{\min}, e.1, e.1)) \\
&= \langle a,2\rangle.(\langle b,2\rangle.1 \oplus \langle c,1\rangle.\langle b,2\rangle.\mathit{Lift}(R_1^{\min}, 1, d.e.1)) \cdot (\langle d,1\rangle. \\
&\quad (\overline{s_2} = e.1{:}\langle e,2\rangle.\mathit{Lift}(R_1^{\min}, 1, 1)) \vee \overline{s_2} = 1{:}\langle e,1\rangle.\mathit{Lift}(R_1^{\min}, 1, 1))) \\
&= \langle a,2\rangle.(\langle b,2\rangle.1 \oplus \langle c,1\rangle.\langle b,2\rangle.1)\cdot(\langle d,1\rangle.([\overline{s_2} = e.1{:}\langle e,2\rangle.1] \vee [\overline{s_2} = 1{:}\langle e,1\rangle].1)),
\end{aligned}
$$

where $\overline{s_2}$ denotes the current state of s_2. The orchestrator inferred by $R_2^{\min}$, including the state-feedback control, is

$$
\begin{aligned}
o_2 = [(s_1 \,|||\, s_2 \,|||\, s_3){:}\langle a,2\rangle].([(s_1 \,|||\, (b.1 + b.e.1) \,|||\, s_3){:}\langle b,2\rangle].1 \oplus \\
[(s_1 \,|||\, (b.1 + b.e.1) \,|||\, s_3){:}\langle c,1\rangle]. \\
[((d.1)\cdot s_1 \,|||\, (b.1 + b.e.1) \,|||\, s_3){:}\langle b,2\rangle].1)\cdot \\
([(s_1 \,|||\, 1 \,|||\, s_3) \vee (s_1 \,|||\, (e.1) \,|||\, s_3) \vee ((d.1)\cdot s_1 \,|||\, 1 \,|||\, s_3) \vee ((d.1)\cdot s_1 \,|||\, (e.1) \,|||\, s_3){:}\langle d,1\rangle]. \\
[(s_1 \,|||\, 1 \,|||\, s_3){:}\langle e,3\rangle].1).
\end{aligned}
$$

In this example, the controller generator is equal to $o_1 \,\square\, o_2$ (see Remark 4.2.6), because there are no iterations in process t. The realization of t results from an

external choice between o_1 and o_2. The use of "$\Box$" emphasizes the role of an oracle in this choice. ■

Example 4.2.3. There is only one possible orchestrator that realizes j on the system described in Example 4.1.1. In the following orchestrator, the state-feedback control is given for all the actions (e.g., the operation *archive* is delegated to s_2 if its current state is $(1+archive)\cdot s_2$, otherwise it is delegated to s_3):[8]

$$
\begin{aligned}
&([\overline{s_2} = s_2{:}\langle write\_story, 2\rangle] \\
&.[\overline{s_2} = (translate + (translate.(1 + archive)))\cdot s_2{:}\langle translate, 2\rangle] \\
&.([\overline{s_2} = s_2{:}\langle archive, 3\rangle].[\overline{s_2} = s_2{:}\langle publish, 3\rangle] \vee [\overline{s_2} = (1+archive)\cdot s_2{:}\langle archive, 2\rangle] \\
&.[\overline{s_2} = s_2{:}\langle publish, 3\rangle]))^*.
\end{aligned}
$$

■

Theorem 4.2.2. Let s and t be a system and a composite component, respectively, such that $t \preceq^{\mathrm{ND}}_{R^{\min}} s$. Let $o = \mathit{Lift}(R^{\min}, t, t)$. Then, there exists a relation $R \subseteq \mathcal{TS}_t \times \mathcal{TS}_o$ such that $t \leftrightarrows^{I_n}_{R} o$.

Proof. Appendix D.1 provides the proof. □

As a corollary, o realizes t on s and $t \simeq^{I_n}_{\mathrm{T}} o$. The intuition behind the theorem comes from the fact that the orchestrator o is embedded in the transition system induced by s. Removing every state q of $\mathcal{TS}_s$ such that there is no state p of $\mathcal{TS}_t$ such that $\langle p, q\rangle \in R^{\min}$ and its ingoing and outgoing transitions leads to a nondeterministic version of o. The deterministic version is obtained from the new operator "$\vee_i [c_i{:}\langle a, k_i\rangle]._$", where the disjuncts refer to all the states of $\mathcal{TS}_s$ in relation with a single state of $\mathcal{TS}_t$ with respect to $R^{\min}$.

8. Only the current state of s_2 appears in the control operators, since $\overline{s_i} = s_i$ for all $i \neq 2$.

4.2.2 Realization of the Control Loop with Synchronization Operators

The control loop depicted in Fig. 3.3 is formalized with the aid of a new ternary process operator "$\rfloor\lfloor$" applied on t, o, and s as follows:

$$t \rfloor o \lfloor s.$$

The current state of s (i.e., $\overline{s}$) is, however, appended to some actions of o and s to perform a step of the control loop in an atomic way. Therefore, $\alpha(o) = \alpha(t) \times I_n \times \mathcal{C}(s, \Sigma_s \cup \{|||\})$ and $\alpha(s) = \big(\cup_i \alpha(s_i)\big) \times I_n \times \mathcal{C}(s, \Sigma_s \cup \{|||\})$.[9] It is assumed that the initial state of the system is observable. The operational semantics of "$\rfloor\lfloor$" is defined by the following rules:

$$\frac{p\!\downarrow, q\!\downarrow, r\!\downarrow}{p \rfloor q \lfloor r\!\downarrow}, \quad \frac{p \xrightarrow{a} p', q \xrightarrow{\langle a,k,r\rangle} q', r \xrightarrow{\langle a,k,r\rangle} r'}{p \rfloor q \lfloor r \xrightarrow{a} p' \rfloor q' \lfloor r'}. \tag{R14}$$

The term $t \rfloor o \lfloor s$ means that t asks the orchestrator for execution of a. Based on the system's current state $\overline{s} = r$, the latter delegates a to s_k for execution. The component s_k performs a and changes state (nondeterministically), which is available for the next step of the loop (this simulates the sending of $\mathtt{ack}(\overline{s_k})$ in Fig. 3.3).[10] In the context of a safety game, the rules R14 indicate the reply issued by the orchestrator as a result of a combined move of the composite component and system.

Example 4.2.4. The orchestrator o_2 in Example 4.2.2 can be rewritten by moving the conditions into the action and replacing the disjunctions by (deterministic) choices. Actions are now 3-tuples (an operation, the index of an available component, a state of the system):

9. Definition 4.2.1 can be generalized to project the traces of actions that belong to $\alpha(o)$ or $\alpha(s)$ onto the traces of actions that, respectively, belong to $\alpha(t)$ or $\big(\cup_i \alpha(s_i)\big)$.

10. In Fig. 3.3, the current state $\overline{s_k}$ is denoted by $\overline{\mathcal{B}_k}$.

$$o_2 = \langle a, 2, s_1 \,|||\, s_2\rangle.(\langle b, 2, s_1 \,|||\, (b.1 + b.e.1)\rangle.1 \oplus \langle c, 1, s_1 \,|||\, (b.1 + b.e.1)\rangle.\langle b, 2, ((d.1)\cdot s_1) \,|||\, (b.1 + b.e.1)\rangle.1)\cdot (\langle d, 1, s_1 \,|||\, 1\rangle \oplus \langle d, 1, s_1 \,|||\, (e.1)\rangle \oplus \langle d, 1, (d.1)\cdot s_1 \,|||\, 1\rangle \oplus \langle d, 1, (d.1\cdot s_1) \,|||\, (e.1)\rangle)\cdot \langle e, 3, s_1 \,|||\, 1\rangle.1.$$

■

The operator "⌋⌊" can be seen as the combination of two other synchronization operators: "⌋" and "⌊". They allow for synchronization between an implementation and the corresponding orchestrator, and between the orchestrator and a system, respectively. The following rules specify their semantics:

$$\frac{p\!\downarrow, q\!\downarrow}{p\rfloor q\!\downarrow}, \quad \frac{p \xrightarrow{a} p', q \xrightarrow{\langle a,k,r\rangle} q'}{p\rfloor q \xrightarrow{a} p'\rfloor q'}, \tag{R15}$$

$$\frac{q\!\downarrow, r\!\downarrow}{q\lfloor r\!\downarrow}, \quad \frac{q \xrightarrow{\langle a,k,r\rangle} q', r \xrightarrow{\langle a,k,r\rangle} r'}{q\lfloor r \xrightarrow{a} q'\lfloor r'}. \tag{R16}$$

Since the alphabets of $t\rfloor o$ and $o\lfloor s$ are the same, the combination is done through the usual synchronous parallel composition $(t\rfloor o)|(o\lfloor s)$, which its semantics is given in Rules R8. Hereafter, the following assumption holds for the next lemmas and the proposition.

Assumption

Let s be a system, t a composite component, and o an orchestrator such that $t \preceq^{\text{ND}}_{R^{\min}} s$ and $o = \mathit{Lift}(R^{\min}, t, t)$.

Lemma 4.2.1. There exists a relation $R' \subseteq \mathcal{TS}_o \times \mathcal{TS}_s$ such that $o \preceq_{R'} s$.

Lemma 4.2.2. There exists a relation $R' \subseteq \mathcal{TS}_o \times \mathcal{TS}_{t\rfloor o\lfloor s}$ such that $o \leftrightarrows^{I_n}_{R'} t\rfloor o\lfloor s$.

Lemma 4.2.3. There exists a relation $R' \subseteq \mathcal{TS}_{t\rfloor o\lfloor s} \times \mathcal{TS}_s$ such that $t\rfloor o\lfloor s \preceq^{I_n}_{R'} s$.

Lemma 4.2.4. There exists a relation $R' \subseteq \mathcal{TS}_o \times \mathcal{TS}_{t\rfloor o}$ such that $o \leftrightarrows^{I_n}_{R'} t\rfloor o$.

Lemma 4.2.5. There exists a relation $R' \subseteq \mathcal{TS}_o \times \mathcal{TS}_{o\lfloor s}$ such that $o \leftrightarrows^{I_n}_{R'} o\lfloor s$.

Lemma 4.2.6. There exists a relation $R' \subseteq \mathcal{TS}_{t\rfloor o} \times \mathcal{TS}_{o\lfloor s}$ such that $t\rfloor o \leftrightarrows_{R'} o\lfloor s$.

Lemma 4.2.7. There exists a relation $R' \subseteq \mathcal{TS}_t \times \mathcal{TS}_{t\rfloor o}$ such that $t \leftrightarrows_{R'} t\rfloor o$.

Lemma 4.2.8. There exists a relation $R' \subseteq \mathcal{TS}_t \times \mathcal{TS}_{t\rfloor o\lfloor s}$ such that $t \leftrightarrows_{R'} t\rfloor o\lfloor s$.

Lemma 4.2.9. There exists a relation $R' \subseteq \mathcal{TS}_{o\lfloor s} \times \mathcal{TS}_s$ such that $o\lfloor s \preceq^{I_n}_{R'} s$.

Lemma 4.2.10. There exists a relation $R' \subseteq \mathcal{TS}_{t\rfloor o} \times \mathcal{TS}_{(t\rfloor o)|(o\lfloor s)}$ such that $t\rfloor o \leftrightarrows_{R'} (t\rfloor o)|(o\lfloor s)$ and there exists a relation $R'' \subseteq \mathcal{TS}_{o\lfloor s} \times \mathcal{TS}_{(t\rfloor o)|(o\lfloor s)}$ such that $o\lfloor s \leftrightarrows_{R''} (t\rfloor o)|(o\lfloor s)$.

Proof. Appendix D.2 provides the proof of all these lemmas. □

Proposition 4.2.1. There exists a relation $R \subseteq \mathcal{TS}_{t\rfloor o\lfloor s} \times \mathcal{TS}_{(t\rfloor o)|(o\lfloor s)}$ such that $t\rfloor o\lfloor s \leftrightarrows_R (t\rfloor o)|(o\lfloor s)$.

Proof. Appendix D.3 provides the proof. □

Figure 4.3 shows the relationships between the processes involved in the control loop depicted in Fig. 3.3. The dashed lines establishes the result of Proposition 4.2.1. Notably, when a reference appears alone, it refers to a lemma.

Rule R14 is too strict when there is not a perfect match between an operation of the composite component and those offered by the available components. If the operations are interpreted as concepts of an ontology, semantic-similarity functions can be defined. For example, let $sim : \alpha(t) \times \left(\cup_i \alpha(s_i) \right) \rightarrow [0, 1]$ be a function that yields the degree of similarity between an operation of the composite component and an operation of an available component (see Def. 5.1.2). The extreme cases $sim(a, a') = 0$ and $sim(a, a') = 1$ mean that a and a' totally differ and perfectly match, respectively. Given a threshold λ, where $0 < \lambda \leq 1$. When the condition $sim(a, a') \geq \lambda$ is satisfied, then a and a' are considered as equivalent (denoted $a \sim a'$). Definition 4.2.3 and Algorithm 2 must, however, be adapted accordingly (see Sect. 5.1.2). Furthermore, the main rule of R14 becomes:

$$\frac{p \xrightarrow{a} p', q \xrightarrow{\langle a', k, r\rangle} q', r \xrightarrow{\langle a', k, r\rangle} r', a \sim a'}{p\rfloor q\lfloor r \xrightarrow{a} p'\rfloor q'\lfloor r'}.$$

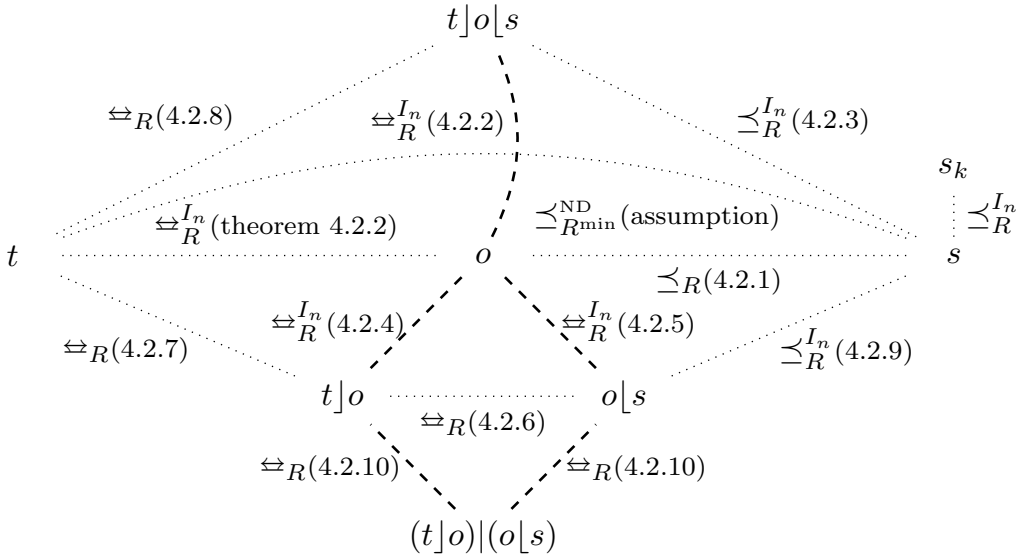


Figure 4.3: A commutative diagram

4.3 Multiple Composite Components

When the dynamic software components are web services, the scenario for control loop, depicted in Fig. 3.3, is too simple to reflect the inherent complexity of web service applications as well as to support modular composition.

Generally, web service applications involve a huge number of composite services (composite components) that invoke both physical and virtual operations in parallel. Typically, the former take effect on remote haptic devices on the IoT while the latter concern acquisition of virtual resources on the cloud or exchange of data with the aim of creating a software resource on the web. These situations give rise to variations and refinements of the scenario depicted in Fig. 3.3.

4.3.1 Multiple Non-Interacting Composite Components

One variation concerns the interleaving execution of multiple non-interacting composite components. It rests on the creation of service (component) instances, implemented through individual threads, for each composite component requesting some of their operations [15]. By spreading the load over possibly multiple cores, this

scenario is particularly adapted to virtual operations. Since physical resources are available for exclusive use only, deadlocks are unavoidable when the amount of concurrent requests on haptic devices increases. In the absence of deadlocks, the only significant source of parallel processing overhead comes from the creation and deletion of threads. Access to the available components (services) is, however, limited by the server resource capacities and traffic rate between the participants. Figure 4.4 shows a sequence diagram for this execution scenario. For a given composite component (e.g., t_1), the sequence of messages exchanged between processes is nearly similar to the one appearing in Fig. 3.3, except for one case. Upon the first reception of a message `delegate` by the master component s_k, the latter creates a thread. The message `CT(`k, i`)` indicates the creation of a thread for component s_k and assigned to the composite component t_i. At this point, the master component transfers its authority to the slave component, which executes the current operation and successive ones without the intermediate of the master component. The frame element labeled with "`alt`" emphasizes these alternatives. Since the requests issued from different composite components and delegated to s_k are processing in parallel, it is embedded in a frame element labeled with "`par`". To avoid overloading the diagram, only the first alternative appears for the composite component t_2.

Definition 4.3.1. Let $s = |||_{k=1}^{n} s_k$, t_i and $o_i = \mathit{Lift}(R^{\min}, t_i, t_i)$ be a system behavior, a composite component, and the orchestrator of t_i, respectively, where $i \in I_m = \{1, \ldots, m\}$ and m is the number of composite components. A thread creation action $\mathrm{CT}(k, i)$ is defined through which a master component s_k creates a slave thread s_{k_i} for t_i. The semantic of this action obeys the following rule:

$$\frac{p_i \xrightarrow{a} p_i', q_i \xrightarrow{\langle a,k,r \rangle} q_i'}{\mathrm{CT}(k,i)}, \; k_i \notin V, \tag{R17}$$

where V is a set that contains the indexes of created threads, namely k_i. This rule states that when t_i asks the orchestrator o_i for the execution of action a and o_i delegates a to s_k, a new slave thread s_{k_i} is created by s_k via the thread creation action.

Upon the creation of slave threads, the system behavior is extended to involve the

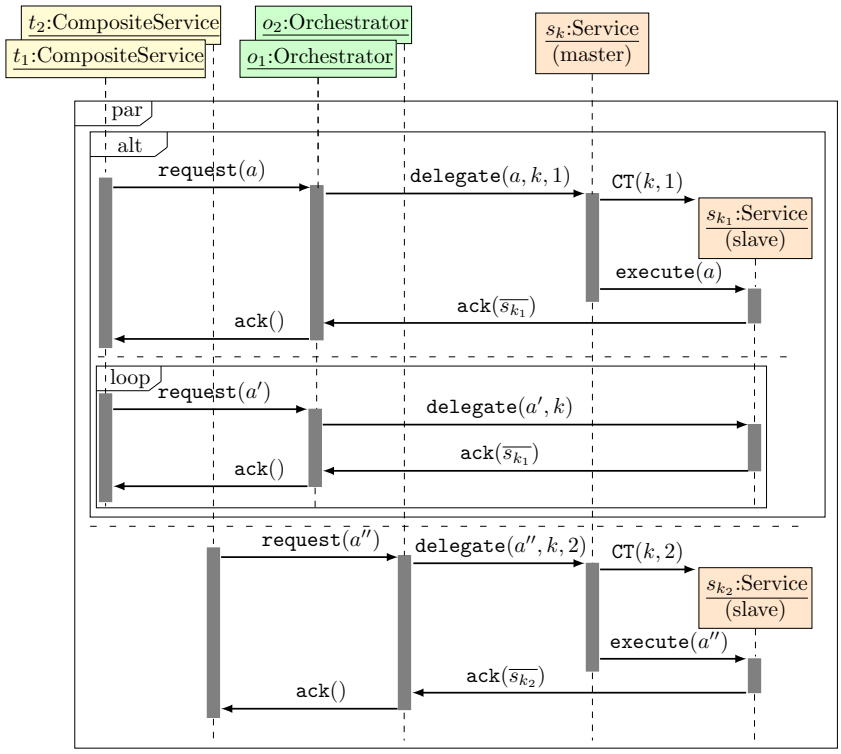


Figure 4.4: Scenario for multiple non-interacting composite components

interleaving of the created threads:

$$s = (|||_{k=1}^{n} s_k) \;|||\; (|||_{k_i \in V} s_{k_i}).$$

Notably, the first term refers to the interleaving of the master components and the second one refers to the interleaving of the slave threads.

Let t_i, $o_i = \mathit{Lift}(R^{\min}, t_i, t_i)$, and s be a composite component, the orchestrator for t_i, and an extended system involving the created threads for t_i ($i \in I_m$), respectively. The control loop depicted in Fig. 4.4 is formalized with the aid of Rules R14 to indicate the interactions between t_i, o_i and their corresponding slave threads $s_{k_i \in V}$ as

follows:

$$\frac{p_i\downarrow, q_i\downarrow, r_{k_i}\downarrow}{p_i\rfloor q_i\lfloor r\downarrow}, \text{ for all } k_i \in V, \tag{R18}$$

$$\frac{p_i \xrightarrow{a} p_i', q_i \xrightarrow{\langle a,k_i,r\rangle} q_i', r \xrightarrow{\langle a,k_i,r\rangle} r'}{p_i\rfloor q_i\lfloor r \xrightarrow{a} p_i'\rfloor q_i'\lfloor r'}, \tag{R19}$$

where $r' = (|||_{j=1}^{n} r_j) \,|||\, (|||_{j_i \in V}\ r'_{j_i})$, and $r'_{j_i} = r_{j_i}$ for $j_i \neq k_i$.

Given the parallel composition (interleaving) of multiple composite components $||_{i\in I_m} t_i$ and multiple orchestrators $||_{i\in I_m} o_i$, both Rules R18 and R19 can be generalized as follows:[11]

$$\frac{||_i p_i\downarrow, ||_i q_i\downarrow, r_{k_i}\downarrow}{||_i(p_i\rfloor q_i\lfloor r)\downarrow}, \text{ for all } k_i \in V \text{ and for all } i \in I_m,$$

$$\frac{||_i p_i \xrightarrow{a} ||_i p_i', ||_i q_i \xrightarrow{\langle a,k_i,r\rangle} ||_i q_i', r \xrightarrow{\langle a,k_i,r\rangle} r'}{||_i(p_i\rfloor q_i\lfloor r) \xrightarrow{a} ||_i(p_i'\rfloor q_i'\lfloor r')}, \text{ for all } i \in I_m,$$

where $p_j' = p_j$, $q_j' = q_j$ for all $j \neq i$, and r' as above.

After the realization of a composite component t_i, the slave threads, which exclusively had interactions with t_i, should be removed from the system s.

Definition 4.3.2. Let $s = (|||_{k=1}^{n} s_k) \,|||\, (|||_{k_i\in V} s_{k_i})$, t_i and $o_i = \mathit{Lift}(R^{\min}, t_i, t_i)$, $i \in I_m$, be a system behavior, a composite component, and its orchestrator, respectively. A remove action $\mathrm{RT}(k,i)$ is defined to eliminate the slave thread s_{k_i}. The semantic for the elimination of all slave threads, already created for t_i, obeys the following rule:

$$\frac{p_i\rfloor q_i\lfloor r\downarrow}{\forall_{k_i\in V}\ \mathrm{RT}(k,i)}. \tag{R20}$$

It means that the remove actions can be executed when the composite component t_i, its orchestrator, and the system are terminated. Notably, the execution of a remove action $\mathrm{RT}(k,i)$ leads to the elimination of the index of s_{k_i} from the set V, namely, $V = V \setminus \{k_i\}$ [15].

11. The parallel composition operator is associative and commutative [56]. It can be generalized to finite combination of processes in the form of $||_i p_i$.

4.3.2 Synchronization on a Unique Component

Other variations emerge when considering modular composition. Indeed, more complex execution scenarios, with their own delegation strategy, become apparent when applications involve more than one composite component with interdependencies between them. For instance, the interdependencies can be expressed as a set of common operations H on which composite components must synchronize. In this case, there are two scenarios: synchronization of multiple composite components on a unique available component and synchronization of multiple composite components on distinct available components. Given the former, when two composite components must synchronize on a common operation $a \in H$, the orchestrator delegates the requested operation to a unique available component that will carry out the operation, particularly on a physical device available in a cloud manufacturing or IoT platform. Figure 4.5 illustrates this scenario, in which the composite component t results from the synchronization of two composite components, namely t_1 and t_2, with respect to H. It is a specialization of the scenario of Fig. 3.3, because t and o have been obtained by synchronization of t_1 and t_2 and o_1 and o_2, respectively. The interface parallel composition operator "$|[\_]|$" is used in that case, which is an associative and commutative operator [56]. Given this operator, the signatures of both composite component t and orchestrator o are, respectively, updated by $\Sigma_t \cup \{|[\_]|\}$ and $\Sigma_o \cup \{|[\_]|\}$.

Let $t = t_1 \,|[H]|\, t_2$ be the result of the interface parallel composition on two composite components. Assume that t is deadlock free and there exists a minimal ND-simulation relation $R^{\min}$ such that $t \preceq^{\mathrm{ND}}_{R^{\min}} s$. An extension in Algorithm 2 with the aim of considering a new case for $p = p_1|[H]|p_2$ leads to obtaining an orchestrator $o = \mathit{Lift}(R^{\min}, t, t)$ in which all common operations of $a \in H$ are delegated to a unique available component. In this case, the algorithm returns $\mathit{Lift}(R^{\min}, p_1, r)|[H]|\mathit{Lift}(R^{\min}, p_2, r)$. Appendix D.4 provides the proof for this case.

Let o_1 and o_2 be orchestrators of t_1 and t_2, respectively, where $o_i = \mathit{Lift}(R_i^{\min}, t_i, t_i)$

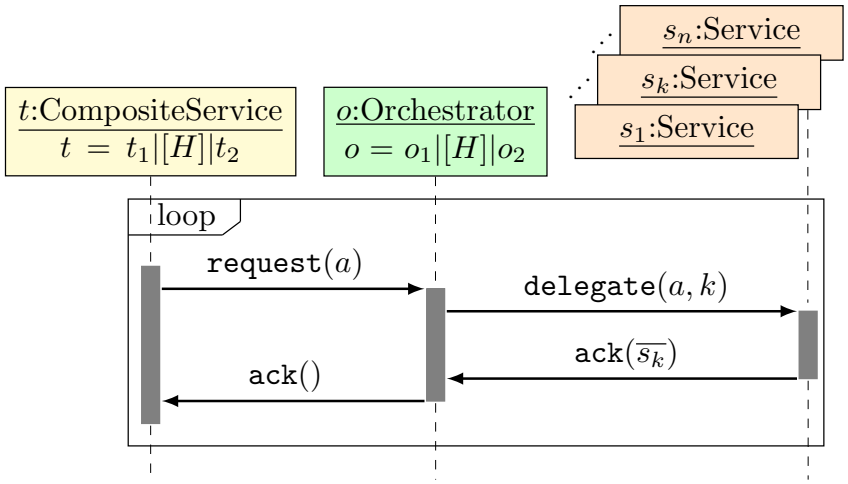


Figure 4.5: Scenario for synchronized composite components—one component

for $i = 1, 2$. The following semantic rules are given for $o_1|[H]|o_2$.

$$\frac{q_1 \xrightarrow{\langle a,k,r\rangle} q_1'}{q_1|[H]|q_2 \xrightarrow{\langle a,k,r\rangle} q_1'|[H]|q_2},\ a \notin H,\quad \frac{q_2 \xrightarrow{\langle a,k,r\rangle} q_2'}{q_1|[H]|q_2 \xrightarrow{\langle a,k,r\rangle} q_1|[H]|q_2'},\ a \notin H,$$
$$\frac{q_1 \xrightarrow{\langle a,k,r\rangle} q_1', q_2 \xrightarrow{\langle a,k,r\rangle} q_2'}{q_1|[H]|q_2 \xrightarrow{\langle a,k,r\rangle} q_1'|[H]|q_2'},\ a \in H. \tag{R21}$$

Let $t_i \preceq^{\mathrm{ND}}_{R_i^{\min}} s$ and $o_i = \mathit{Lift}(R_i^{\min}, t_i, t_i)$ for $i \in I_m = \{1, \ldots, m\}$, where m is the number of composite components. In this case, it is possible that $|[H]|_i\, o_i$ exhibits deadlock even if $|[H]|_i t_i$ is deadlock free. On the synchronization of operation $a \in H$, it is impossible that the operation be delegated to the same available component by all the orchestrators. The following condition should hold for $|[H]|_i\, o_i$ to be deadlock free.

- $|[H]|_i t_i$ is deadlock free and
- for all $a \in H$ such that $|[H]|_i\, p_i \xrightarrow{a} |[H]|_i\, p_i'$ $(p_i, p_i' \in \mathcal{C}(t_i, \Sigma_{t_i}))$, then $q_i \xrightarrow{\langle a,k,r\rangle} q_i'$ $(q_i, q_i' \in \mathcal{C}(o_i, \Sigma_{o_i}))$, $p_i \leftrightarrows^{I_n}_{R_i} q_i$ and $p_i' \leftrightarrows^{I_n}_{R_i} q_i'$) for all $i \in I_m$.

The first subcondition implies that $|[H]|_i\, o_i$ is deadlock free. Operations are, however, delegated independently to available components with potential conflicts unless available components support multithreading. The second subcondition forces or-

chestrators to delegate the operations (on which composite components synchronize) to the same available component. So, $|[H]|_i\, q_i \xrightarrow{\langle a,k,r\rangle} |[H]|_i\, q_i'$ by Rule R21.

Let $s = |||_{k=1}^{n} s_k$ be a system behavior. Moreover, let $t = |[H]|_i\, t_i$ and $o = |[H]|_i\, o_i$ with a common operation set H be, respectively, the results of the interface parallel composition on multiple composite components and multiple orchestrators, where $i \in I_m$ and o is deadlock free. Besides, for each t_i, there exists an o_i such that $o_i = \mathit{Lift}(R^{\min}, t_i, t_i)$. The process term $|[H]|_i\, t_i \rfloor |[H]|_i\, o_i \lfloor s$ is the one for the control loop depicted in Fig 4.5 and obeys Rules R14.

Example 4.3.1. If, in the Example 4.1.1, the composite components are synchronized on the operation *archive*, then there is no orchestrator that realizes $j|[\{\mathit{archive}\}]|r$. This is due to the delegation of *archive* to two different services (components) (to s_2 or s_3 for j and s_1 for r). ■

4.3.3 Synchronization on Distinct Components

In this scenario, as in the synchronization on a unique component, the synchronization is still maintained, but the common operation $a \in H$ is delegated to two distinct available components for execution. The scenario is particularly attractive when the goal of a composite component is to coordinate operations of multiple haptic devices almost simultaneously when people participate to a social activity. In general, there are as many components that carry out the operation as there are composite components involved in the synchronization. Figure 4.6 illustrates this scenario, in which the composite component t results from the synchronization of two composite components, namely t_1 and t_2, on distinct available components s_j and s_k with respect to H.

For the aim of synchronization in this scenario, a new operator $\big[[H]\big]$, called *distinct synchronization*, is defined on orchestrators. For two different orchestrators o_1 and o_2, the semantics rules of $o_1 \big[[H]\big] o_2$ are given as follows:

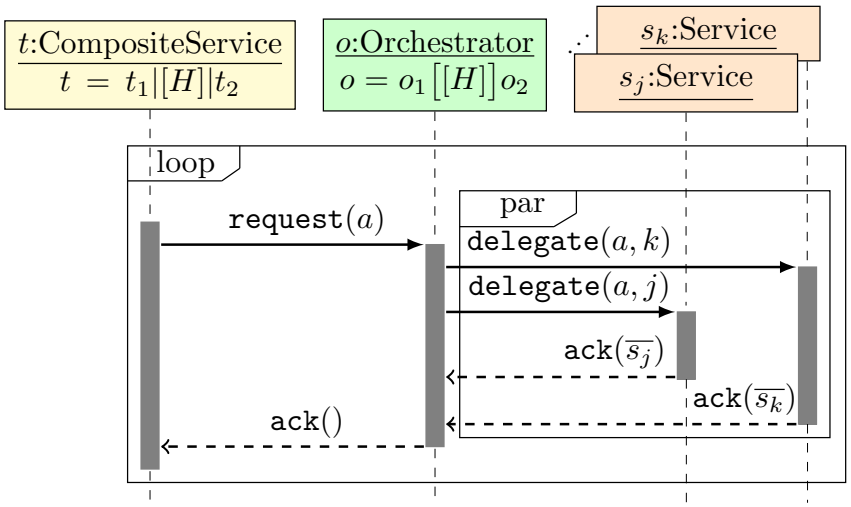


Figure 4.6: Scenario for synchronized composite components—two components

$$\frac{p \xrightarrow{\langle a,\{k\},r\rangle} p'}{p\big[[H]\big]q \xrightarrow{\langle a,\{k\},r\rangle} p'\big[[H]\big]q}, a \notin H, \quad \frac{q \xrightarrow{\langle a,\{k\},r\rangle} q'}{p\big[[H]\big]q \xrightarrow{\langle a,\{k\},r\rangle} p\big[[H]\big]q'}, a \notin H,$$
$$\frac{p \xrightarrow{\langle a,K,r\rangle} p', q \xrightarrow{\langle a,J,r\rangle} q', K \cap J = \emptyset}{p\big[[H]\big]q \xrightarrow{\langle a,K\cup J,r\rangle} p'\big[[H]\big]q'}, a \in H, \tag{R22}$$

where K and J are subsets of I_n.

Since all composite components t_i ($i \in I_m$) should be synchronized on distinct available components with respect to $a \in H$, a function is defined as below to map each index of a composite component to the index of an available component that executes a.

Definition 4.3.3. Let $I_m = \{1, \ldots, m\}$ and $I_n = \{1, \ldots, n\}$ be two sets of indexes, where $|I_m| \leq |I_n|$. An injective function $\Gamma : I_m \to I_n$ is defined to map each index $i \in I_m$ to an index $k \in I_n$.

Let $t_i \preceq^{\mathrm{ND}}_{R_i^{\min}} s$ and $o_i = \mathit{Lift}(R_i^{\min}, t_i, t_i)$ for $i \in I_m = \{1, \ldots, m\}$, where m is the number of composite components. In this case, it is possible that $\big[[H]\big]_i o_i$ exhibits

deadlock even if $|[H]|_i\, t_i$ is deadlock free. On the synchronization of operation $a \in H$, it is impossible that the operation be delegated to distinct available components by all the orchestrators. The following condition should hold for $\big[[H]\big]_i o_i$ to be deadlock free.

- $|[H]|_i t_i$ is deadlock free and
- for all $a \in H$ such that $|[H]|_i\, p_i \xrightarrow{a} |[H]|_i\, p_i'$ $(p_i, p_i' \in \mathcal{C}(t_i, \Sigma_{t_i}))$, then $q_i \xrightarrow{\langle a, \{\Gamma(i)\}, r\rangle} q_i'$ $(q_i, q_i' \in \mathcal{C}(o_i, \Sigma_{o_i})$, $p_i \leftrightarrows_{R_i}^{I_n} q_i$ and $p_i' \leftrightarrows_{R_i}^{I_n} q_i')$ for all $i \in I_m$.

The first subcondition implies that $\big[[H]\big]_i\, o_i$ is deadlock free. Again, operations are, however, delegated independently to available components with potential conflicts unless available components support multithreading. The second subcondition forces orchestrators to delegate the operations (on which composite components synchronize) to the distinct available components. So, $\big[[H]\big]_i\, q_i \xrightarrow{\langle a, \cup_i\{\Gamma(i)\}, r\rangle} \big[[H]\big]_i\, q_i'$ by Rule R22.

Let $s = |||_k s_k$, $k \in I_n$, be a system behavior, $t = |[H]|_i\, t_i$ and $o = \big[[H]\big]_i\, o_i$, $i \in I_m$, be the result of the interface parallel composition on multiple composite components that is deadlock free and the result of distinct synchronization operator on multiple orchestrators, respectively, where $o_i = \mathit{Lift}(R^{\min}, t_i, t_i)$, H is a common operation set, and o is deadlock free. In a sense, for each operation $\langle a, k, r\rangle \in \alpha(o_i)$, where $a \in H$, there is a map between k and i, namely $\Gamma(i) = k$. So, the control loop depicted in Fig. 4.6 with the aid of the main rule R14 is formalized as follows.

$$\frac{p \xrightarrow{a} p', q \xrightarrow{\langle a, \cup_i\{\Gamma(i)\}, r\rangle} q', r \xrightarrow{\langle a, \cup_i\{\Gamma(i)\}, r\rangle} r'}{p \rfloor q \lfloor r \xrightarrow{a} p' \rfloor q' \lfloor r'}, \; a \in H \text{ and for all } i \in I_m,$$

where $p = |[H]|_i\, p_i$, $p' = |[H]|_i\, p_i'$, $q = \big[[H]\big]_i\, q_i$, $q' = \big[[H]\big]_i\, q_i'$, $r' = |||_{k=1}^{n} r_k'$ with $r_k' = r_k$ for all $k \notin \cup_i\{\Gamma(i)\}$.

It should be noted that, for $a \notin H$, the semantic rule for this scenario obeys exactly the main rule of R14.

Example 4.3.2. If, in the Example 4.1.1, the composite components synchronize on the operation *translate*, then there is an orchestrator $o = o_1 \big[[\mathit{translate}]\big] o_2$ that realizes

$t = j|[\{translate\}]|r$. This is due to the delegation of *translate* to two different services (components) (to s_2 for j and s_4 for r). ■

4.3.4 Phenomenon of Deadlock

In the last two scenarios, when the common operation set is empty, there can be an interleaving among multiple composite components. Besides, each composite component through its orchestrator interacts with a shared system s, whose operations are exclusive. Assuming that, the available components of the system have the potential of locking their operations to provide an exclusive access for each composite component. If the composite components through their orchestrators have a competition to access such operations, then the phenomenon of deadlock is possible. Before providing a formal definition for such a phenomenon, the notions of lock, unlock, and block should be formally defined.

Each available component is able to lock and unlock its operation through defining two actions `Lock` and `Unlock` [15]. The formal definitions of such actions along with their semantics rules are presented as follows.

Definition 4.3.4. Let t_i, $o_i = \mathit{Lift}(R^{\min}, t_i, t_i)$, and $s = |||_{k=1}^{n} s_k$ be a composite component, an orchestrator, and a system behavior, where $i \in I_m = \{1, \ldots, m\}$ and m is the number of composite components. An action $\texttt{Lock}(\langle a, k, r\rangle, t_i)$ is defined to lock an operation $\langle a, k, r\rangle \in \alpha(s_k)$ to be exclusively accessible by the composite component t_i. The semantics rule of this action is given as follows:

$$\frac{p_i \xrightarrow{a} p_i', q_i \xrightarrow{\langle a,k,r\rangle} q_i'}{\texttt{Lock}\ (\langle a, k, r\rangle, p_i)}. \tag{R23}$$

It means that when a requested operation a of t_i is delegated to s_k by o_i, s_k locks $\langle a, k, r\rangle$ for t_i.

Definition 4.3.5. Let t_i be a composite component and $s_k \in s$ be an available component. An action $\texttt{Unlock}(\langle a, k, r\rangle, t_i)$ is defined to release an operation $\langle a, k, r\rangle \in \alpha(s_k)$ already locked for t_i. The semantics rule of this action is given as follows:

$$\frac{p_i \rfloor q_i \lfloor r\downarrow}{\forall_{\langle a,k,r\rangle \in Lock_i} \texttt{Unlock}\ (\langle a, k, r\rangle, p_i)}, \tag{R24}$$

where $Lock_i$ be a set, which contains the operations $\langle a, k, r\rangle$ in system that were locked for a composite component t_i. This rule states when the composite component t_i, its orchestrator o_i, and the system s terminate, the unlock actions are activated to release all operations, already locked for t_i.

Definition 4.3.6. Let t_i and t_j, where $i \neq j$, be two composite components and $o_i = \mathit{Lift}(R^{\min}, t_i, t_i)$ and $o_j = \mathit{Lift}(R^{\min}, t_j, t_j)$ be their orchestrators, respectively. Furthermore, let $Lock_i$ and $Lock_j$ be the sets of locked operations for t_i and t_j, respectively. The process $t_j \rfloor o_j$ blocks $t_i \rfloor o_i$, if o_i tries to delegate an operation $\langle a, k, r\rangle$ to s_k such that $\langle a, k, r\rangle \in Lock_j$. It is said that $t_i \rfloor o_i$ becomes a blocked process, denoted $Block_i^j$, if $t_i \rfloor o_i$ is blocked by $t_j \rfloor o_j$. The following semantic rule is given for a blocked process:

$$\frac{p_i \xrightarrow{a} p_i', q_i \xrightarrow{\langle a,k,r\rangle} q_i'}{(p_i \rfloor q_i)||(p_j \rfloor q_j) \xrightarrow{a} Block_i^j||(p_j \rfloor q_j)}, \langle a, k, r\rangle \in Lock_j \text{ and } i \neq j.$$

Definition 4.3.7. The interleaving of $t_i \rfloor o_i$ and $t_j \rfloor o_j$ leads to a deadlock, denoted δ, if the processes $t_i \rfloor o_i$ and $t_j \rfloor o_j$ have been blocked by each other. The semantics of deadlock obeys the following rule:

$$\frac{Block_i^j || Block_j^i}{\delta}, i \neq j.$$

Example 4.3.3. It is supposed that in Example 4.1.1, there are two new composite components $t_1 = j \cdot r$ and $t_2 = r \cdot j$, which are realized by the orchestrators $o_1 = \mathit{Lift}(R^{\min}, t_1, t_1)$ and $o_2 = \mathit{Lift}(R^{\min}, t_2, t_2)$. Imagine that the operations $\langle \mathit{upload\_photo}, 4, \overline{s_4}\rangle \in \alpha(s_4)$ and $\langle \mathit{publish}, 3, \overline{s_3}\rangle \in \alpha(s_3)$ are exclusive. The interleaving $(t_1 \rfloor o_1)||(t_2 \rfloor o_2)$ with the assumptions of $\langle \mathit{upload\_photo}, 4, \overline{s_4}\rangle \in Lock_2$ and $\langle \mathit{publish}, 3, \overline{s_3}\rangle \in Lock_1$ can lead to a deadlock, since given the order of operations in t_1 and t_2, $t_1 \rfloor o_1$ exhibits $Block_1^2$ through the request of $\langle \mathit{upload\_photo}, 4, \overline{s_4}\rangle$. Likewise, $t_2 \rfloor o_2$ exhibits $Block_2^1$ through the request of $\langle \mathit{publish}, 3, \overline{s_3}\rangle$.

■

The aforementioned definitions can be used in future research especially for the case of deadlock freedom strategies as those described in [15].

4.4 Modular Control

In modular control, the combination of components is done with algebraic process expressions. The operators "$\cdot$", "$\oplus$", "*", and "$|[\_]|$" are used in such expressions. The following example motivates modular control of components. It shows how it is advantageous to reuse computed orchestrators to control a combined component.

Example 4.4.1. A multimedia reporter can be obtained by the combination of a traditionalist journalist (represented by j in Example 4.1.1) and a web video reporter (represented by r in Example 4.1.1). The process term $(r \oplus j)^*$ represents its behavior and $(o_j \oplus o_r)^*$ its orchestrator, where o_j is the orchestrator for j and o_r is one of the three possible orchestrators for r. ■

The previous example raises several important questions. Let o_1 and o_2 be two orchestrators that realize t_1 and t_2, respectively, on a system s. Let $t_1 \odot t_2$ be a process expression, where $\odot \in \{\cdot, \oplus, ^*, |[\_]|\}$. Under what conditions can t_1 and t_2 be combined? Is $o_1 \odot o_2$ a realization of $t_1 \odot t_2$? Is there a relation $R^{\min}$ such that $t_1 \odot t_2 \preceq^{\mathrm{ND}}_{R^{\min}} s$, if $t_i \preceq^{\mathrm{ND}}_{R_i^{\min}} s$ $(i = 1, 2)$? What is the relationship between $o_1 \odot o_2$ and $\mathit{Lift}(R^{\min}, t_1 \odot t_2, t_1 \odot t_2)$? The first question is related to the fact that an implementation must be deterministic and terminates (or *well-formed* hereafter). The second question depends on how many available components can be allowed to execute a common operation (when $\odot$ is $|[\_]|$). The third question is answered by examining the structure of available components. The last question is particularly interesting when utility values (e.g., cost, quality of service) are associated with orchestrators or similar operations, which are considered equivalent, are authorized.

Definition 4.4.1. Let $\mathbb{T}(\mathcal{TS}_p)$ be the traces of the transition system induced from p. The notation $\alpha_1(p)$ is defined to denote the set $\{w_0 \mid w \in \mathbb{T}(\mathcal{TS}_p)\} \subseteq \alpha(p)$, where w_0 is the head (first action) of the trace w.

Definition 4.4.2. The notation $\alpha_{\downarrow}(p)$ is defined to denote the set of operations involved in a choice offered by p with process 1.

Example 4.4.2. Given the process s_3 in Example 4.2.1, $\alpha_{\downarrow}(s_3) = \{e\}$. Furthermore, based on the trace $e.1 \in \mathbb{T}(\mathcal{TS}_{s_3})$, $\alpha_1(s_3) = \{e\}$. ■

Lemma 4.4.1. Let t, t_1, and t_2 be well-formed process terms defined from the signature Σ_t. Then:

- $t_1 \cdot t_2$ is well-formed if $\alpha_{\downarrow}(t_1) \cap \alpha_1(t_2) = \emptyset$;
- $t_1 \oplus t_2$ is well-formed if $\alpha_1(t_1) \cap \alpha_1(t_2) = \emptyset$;
- t^* is well-formed;
- $t_1 | [H] | t_2$ is well-formed if $H \subseteq \alpha(t_1) \cap \alpha(t_2)$ and no deadlock.

Proof. Appendix D.5 provides the proof. □

Checking the deadlock condition associated with $t_1|[H]|t_2$ in Lemma 4.4.1 has essentially the same computational complexity as the calculation of an orchestrator by the monolithic approach. This is not really surprising. The same situation occurs when checking the nonconflicting property to ensure nonblocking modular solution in supervisory control [20].

Example 4.4.3. The process terms $1 \oplus (a.(1 \oplus b) \cdot 1)$ and $b.1$ cannot be concatenated because $\alpha_{\downarrow}(1 \oplus (a.(1 \oplus b) \cdot 1)) = \{a, b\}$ and $\alpha_1(b.1) = \{b\}$. The process terms $a.b.1$ and $a.c.1$ cannot be combined by using the operator "$\oplus$" because $\alpha_1(a.b.1) = \alpha_1(a.c.1) = \{a\}$. Otherwise, nondeterminism is introduced. ■

Proposition 4.4.1. Let t_1 and t_2 be process terms over Σ_t representing the implementation parts of two components such that $t_i \preceq^{\text{ND}}_{R_i^{\min}} s$, $(i = 1, 2)$. Let o_1 and o_2 be such that $o_i = \mathit{Lift}(R_i^{\min}, t_i, t_i)$, $(i = 1, 2)$. Under the second condition of Lemma 4.4.1, $o_1 \oplus o_2$ realizes $t_1 \oplus t_2$, in particular there exists a minimal ND-simulation relation $R^{\min}$ such that $t_1 \oplus t_2 \preceq^{\text{ND}}_{R^{\min}} s$, $o = \mathit{Lift}(R^{\min}, t_1 \oplus t_2, t_1 \oplus t_2)$.

Proof. Appendix D.6 provides the proof. □

Remark 4.4.1. Let n_i be the number of $R_i^{\min}$ (or o_i) such that $t_i \preceq^{\text{ND}}_{R_i^{\min}} s$, $(i = 1, 2)$, then there are $n_1 \times n_2$ combinations of two orchestrators under the operator "$\oplus$" and the same number of individual orchestrators for $t_1 \oplus t_2$.

For the next two propositions, it is assumed that all operations delegated to s_i are the only ones that appear on transitions of a strongly connected component of $\mathcal{TS}_{s_i}$ (or a self-loop if there is only one operation), for all $i \in I_n$.

Proposition 4.4.2. Let t be process terms over Σ_t representing the implementation part of a component such that $t \preceq^{\text{ND}}_{R^{\min}} s$. Let o be such that $o = \mathit{Lift}(R^{\min}, t, t)$. Under the third condition of Lemma 4.4.1 and the above assumption, o^* realizes t^*, in particular, there exists a minimal ND-simulation $R'^{\min}$ such that $t^* \preceq^{\text{ND}}_{R'^{\min}} s$, $o' = \mathit{Lift}(R'^{\min}, t^*, t^*)$.

Proof. Appendix D.7 provides the proof. □

Remark 4.4.2. Let n be the number of orchestrators for t. Then, there are the same number of individual orchestrators for t^* due to the aforementioned assumption on all operations delegated to an available component.

Proposition 4.4.3. Let t_1 and t_2 be process terms over Σ_t representing the implementation parts of two components such that $t_i \preceq^{\text{ND}}_{R_i^{\min}} s$, $(i = 1, 2)$. Let o_1 and o_2 be such that $o_i = \mathit{Lift}(R_i^{\min}, t_i, t_i)$, $(i = 1, 2)$. Under first condition of Lemma 4.4.1 and the above assumption, $o_1 \cdot o_2$ realizes $t_1 \cdot t_2$, in particular, there exists a minimal ND-simulation relation $R^{\min}$ such that $t_1 \cdot t_2 \preceq^{\text{ND}}_{R^{\min}} s$, $o = \mathit{Lift}(R^{\min}, t_1 \cdot t_2, t_1 \cdot t_2)$.

Proof. Appendix D.8 provides the proof. □

Remark 4.4.3. Let n_i be the number of $R_i^{\min}$ (or o_i) such that $t_i \preceq^{\text{ND}}_{R_i^{\min}} s$, $(i = 1, 2)$, then there are $n_1 \times n_2$ combinations of two orchestrators under the operator "$\cdot$". The number of individual orchestrators for $t_1 \cdot t_2$ can be $n_1 \times n_2 + n_3$, where n_3 is the number of possible minimal ND-simulation relations $R^{\min}$ with following condition:

$$R^{\min} = \{\langle p_1 \cdot p_2, q\rangle \mid \exists R_1^{\min} \text{ with } \langle p_1 \cdot p_2, q\rangle \in R_1^{\min} \text{ and} \\ \text{for } p_1\!\downarrow\colon \nexists R_2^{\min} \text{ with } \langle p_1 \cdot p_2, q\rangle \in R_2^{\min}\},$$

where $p_1 \in \mathcal{C}(t_1, \Sigma_t), p_2 \in \mathcal{C}(t_2, \Sigma_t)$ and $q \in \mathcal{C}(s, \Sigma_s \cup \{\|\|\})$.

Proposition 4.4.3, could be generalized to the m-fold sequential composition (see [6]) the additional assumption on structures $\mathcal{TS}_{s_i}$ could be relaxed accordingly.

Example 4.4.4. Let $t = a.1$ and $s = a.1 + 1$. Then, there exists $R^{\min}$ such that $t \preceq^{\text{ND}}_{R^{\min}} s$, but there exists no $R^{\min}$ such that $t^* \preceq^{\text{ND}}_{R^{\min}} s$. The conclusion is the same with $t \cdot t$, because s can only perform the operation a once. This is, however, not the case if $s = a^*$. ■

Proposition 4.4.4. Let t_1 and t_2 be process terms over Σ_t representing the implementation parts of two components such that $t_i \preceq^{\mathrm{ND}}_{R_i^{\min}} s$, $(i = 1, 2)$. Let o_1 and o_2 be such that $o_i = \mathit{Lift}(R_i^{\min}, t_i, t_i)$, $(i = 1, 2)$. Under the last condition of Lemma 4.4.1 and the condition that $o_1|[H]|o_2$ is not deadlock, $o_1|[H]|o_2$ realizes $t_1|[H]|t_2$, in particular, there exists a minimal ND-simulation relation $R^{\min}$ such that $t_1|[H]|t_2 \preceq^{\mathrm{ND}}_{R^{\min}} s$, $o = \mathit{Lift}(R^{\min}, t_1|[H]|t_2, t_1|[H]|t_2)$.

Proof. Appendix D.9 provides the proof. □

Remark 4.4.4. Let n be the number of combinations of two orchestrators under the operator "$|[\_]|$". The number of individual orchestrators for $t_1|[H]|t_2$ can be $n + m$, where m is the number of possible minimal ND-simulation relations $R^{\min}$ with following condition:

$$R^{\min} = \{\langle p_1|[H]|p_2, q\rangle \mid \exists R_1^{\min} \text{ with } \langle p_1|[H]|p_2, q\rangle \in R_1^{\min} \text{ if and only if } \nexists R_2^{\min} \text{ with } \langle p_1|[H]|p_2, q\rangle \in R_2^{\min}\},$$

where $p_1 \in \mathcal{C}(t_1, \Sigma_t), p_2 \in \mathcal{C}(t_2, \Sigma_t)$ and $q \in \mathcal{C}(s, \Sigma_s \cup \{|||\})$.

Remark 4.4.5. If a cost is assigned to every component, then combining reusable orchestrators can lead to a nonoptimal solution when two components are combined under the operator $\odot \in \{\cdot, |[\_]|\}$. In fact, it is possible to have more orchestrators o for $t_1 \odot t_2$ (n_3 and m, respectively) than combinations of orchestrators o_1 and o_2 for t_1 and t_2. In such cases, it is possible that $\mathit{cost}(o) < \mathit{cost}(o_1) + \mathit{cost}(o_2)$.

4.5 Contributions

In summary, this chapter makes the following contributions.

- It reformulates the main elements of the original behavior composition framework as process terms with the aid of a process calculus.
- It defines an index-less bisimulation relation to check whether the implementation of a composite component and its corresponding orchestrator are bisimilar.
- It introduces a new control operator with its semantic rule in order to obtain deterministic orchestrators.

– It proposes an algorithm to obtain the process term for the orchestrator from a minimal ND-simulation relation.

– It introduces a ternary process operator with new semantic rules for the synchronization among an implementation, its corresponding orchestrator, and a system. Besides, the main semantic rule of this operator is extended to consider the similarity between the operations of the implementation and system.

– It provides the interleaving execution of multiple composite components, including multithreading, synchronization on a unique available component, and synchronization on distinct available components. Besides, it introduces some actions and semantic rules to formally define deadlock detection.

– It constitutes a modular control through which precomputed orchestrators are combined using process operators to realize a new complex implementation.

– It establishes the relationships between the orchestrator of combined implementations and the combined orchestrators of the corresponding implementations.

Chapter 5

Semantic-Supported and Preference-Based Composition

In large scale computing systems, analogous to cloud computing, a huge number of computing resources as dynamic software components are offered by many cloud providers and brokers. In some cases, a user may have no information about the exact name of a resource and demands it to a cloud broker. In this case, the broker should offer similar resources that have the same functionality with the one requested by the user. Moreover, a user may express some preferences for the requested resources.

In cloud computing, each broker can determine a behavior for the execution order of its resources. When such a behavior is modeled by a finite state machine, the behavior composition framework can be integrated into the cloud to provide an automatic mechanism for resource composition. However, such an integration requires several extensions in the original framework, which consists in three phases summarized as follows.

- The adoption of a semantic-based framework in which a requested action (resource) and an action in an available broker that have different names but the same functionality are considered similar. This phase focuses on the notions of ontology and resource reasoning to define similar actions. The proposed solution revisits the largest ND-simulation relation as defined in Sect. 2.2 to match the similar actions.
- The adoption of a preference-based framework which supports both qualitative and quantitative preferences exposed by a target behavior. Multiple attributes are assigned to the actions of available behaviors (brokers) and some preferences are attached to the requested actions of the target behavior. The proposed solution revisits again the largest ND-simulation relation as defined in Sect. 2.2 to match the values of such attributes with preferences of the target.

– The development of a hybrid framework that combines semantic-based and preference-based frameworks. In this end, the proposed solution revisits the largest ND-simulation relation of the preference-based framework not only to match similar actions, but also to match the preferences and the attribute values that are similar.

5.1 Reasoning Based on Semantics

To provide a more flexible framework for behavior composition, it was suggested to introduce a compatibility relation $\ll \subseteq A \times A$ over the set of actions [32]. It substitutes for the present equality between actions in the definition of the largest ND-simulation relation and the underlying algorithm that computes it (see Sect. 2.2). An action a' can now be carried out by an available behavior, if it is compatible with the delegated action a, that is, $a \ll a'$. No more details were given about this issue by the authors.

The use of resource reasoning metrics constitutes a first appealing solution [3, 57]. It evaluates the degree of match between any two resources. Generally, such metrics fit with a domain ontology graph, which has a well-formed structure to determine the multipaths connecting two concepts. Hence, the main effort must be concentrated on building ontologies.

5.1.1 Ontology and Resource Reasoning

Ontology is a representational artifact whose purpose is the exhibition of entities, defined classes, and relations between them [4]. An ontology can offer meta-information to describe semantics of data and allows for building knowledge bases. Furthermore, it is a formal structure that supports the communication between a user and a computer agent [3].

The kinds of ontology are classified into domain ontology, reference ontology, top-level ontology, and application ontology [4]. The intended class of ontology for better representation and classification of resources included in a specific system is the domain one. It provides a taxonomy with a hierarchical structure for such resources, considered as concepts, together with a set of axioms identifying several rules to show

how the concepts and relations can be comprehended [4]. A typical example is a cloud ontology which supplies a taxonomy for its computing resources [57].

Different kinds of relations can be defined between the concepts in a domain ontology. For instance, "*is-a*", "*part-of*", "*is-subtype-of*", "*is-member-of*", "*participates-in*", "*has-output*", and "*precedes*" are some examples of such relations. A domain ontology is formally defined as follows.

Definition 5.1.1. [34] An ontology in a specific domain O is a tuple of $\langle C, \leq c, R, \leq r, A\rangle$, where C is a set of concepts, R is a set of relations, $\leq c$ is a partial order on C that is called the concept hierarchy, $\leq r$ is a partial order on R that is called the relation hierarchy, and A is a set of axioms including rules in the logical forms to describe the relationships among the concepts. [1]

Based on a domain ontology, a graph, called an ontology graph, is drawn to demonstrate a taxonomy [3]. In this graph, each concept is represented as a node and each edge indicates a relationship between two concepts. More precisely, each edge illustrates a relation such as "*is-a*" or "*part-of*". For example, Fig. 5.1 depicts a simple ontology graph for a web agency providing travel services. In this graph, the *Travel agency services* is considered as the root node having subnodes including *Accommodation reservation*, *Transportation reservation*, and *Meal reservation*.

In cloud computing, where resources are defined semantically, the notion of resource reasoning is put forward, which includes similarity, compatibility, and numerical reasoning [57]. In similarity reasoning, to measure the degree of similarity between two different concepts, several semantic similarity functions have been introduced. Among those proposed in [3, 49, 54], there is one that defines a function being compatible with a hierarchical structure of well-formed concepts that can be found in a domain ontology graph [3]. This function takes into account specialization or generalization of one concept with respect to another.

Definition 5.1.2. [3] The semantic similarity function $sim : C \times C \rightarrow [0, 1]$ is defined as:

$$sim(x, y) = \rho \frac{|\alpha(x) \cap \alpha(y)|}{|\alpha(x)|} + (1 - \rho) \frac{|\alpha(x) \cap \alpha(y)|}{|\alpha(y)|}, \tag{5.1.1}$$

1. The notations $\leq c$ and $\leq r$ could be replaced by $\leq_c$ and $\leq_r$, respectively. Although the latter are better, the former were adopted to avoid confusion with [34].

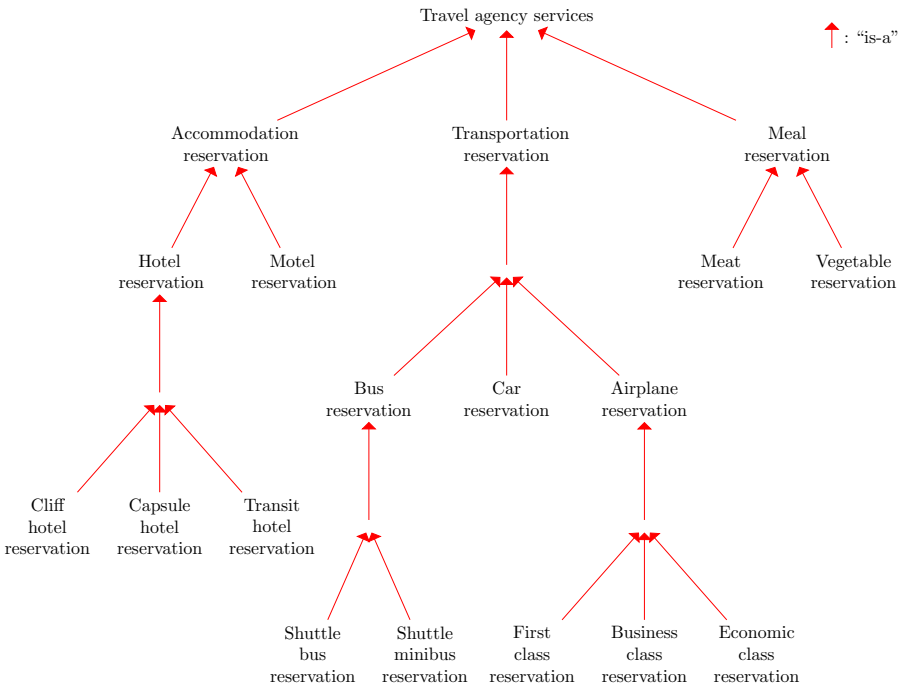


Figure 5.1: A part of simple ontology graph of a travel agency service

where the constant $\rho \in [0, 1]$ determines the degree of influence of generalization, the term $\alpha(x)$ is considered as the set of upward nodes reachable from x (including the node labeled by x), and the expression $\alpha(x) \cap \alpha(y)$ is the reachable common nodes between x and y.

For instance, in Fig. 5.1, the concept of *Meal reservation* has three reachable upward nodes from itself, whereas this is four for *Cliff hotel reservation*. Hence,

$$|\alpha(\textit{Meal reservation})| = 3 \quad \textit{and} \quad |\alpha(\textit{Cliff hotel reservation})| = 4.$$

In addition, the number of common nodes for *Cliff hotel reservation* and *Transit hotel reservation* is more than *Cliff hotel reservation* and *Bus reservation*, which are calculated as follows:

$$|\alpha(\textit{Cliff hotel reservation}) \cap \alpha(\textit{Transit hotel reservation})| = 3 \text{ and}$$
$$|\alpha(\textit{Cliff hotel reservation}) \cap \alpha(\textit{Bus reservation})| = 1.$$

As indicated in Eq. 5.1.1, the semantic similarity function maps two concepts into the unit interval, and its output shows the degree of similarity between x and y. So, the value 0 means no similarity and 1 means full similarity.

Given the ontology graph and semantic similarity function, a square matrix of order n of similarities among concepts is constructed:

$$SIM = \begin{bmatrix} sim(a_1, a_1) & \dots & sim(a_1, a_n) \\ \vdots & \ddots & \vdots \\ sim(a_n, a_1) & \dots & sim(a_n, a_n) \end{bmatrix},$$

where n is the number of concepts. It is important to note that each element in this matrix indicates a real number in [0,1] giving the degree of similarity between related concepts. Moreover, a threshold in the interval (0,1] is defined to accept the minimum measure of similarity between two concepts.

The similarity reasoning was introduced to measure the degree of similarity for functional requirements with the aid of a semantic similarity function (see Def. 5.1.2). To calculate the degree of match for technical requirements in computing systems, both the compatibility and numerical reasoning were proposed.

The compatibility reasoning is appropriate for comparing two sibling nodes in a domain ontology graph, for example, the compatibility between two different versions of a software program in the cloud ontology.

Definition 5.1.3. [57] The compatibility reasoning function $compat : C \times C \to (0, 2)$ is defined as:

$$compat(x, y) = sim(x, y) + \frac{\mu^{|c_x - c_y|}}{\theta}, \tag{5.1.2}$$

where $0 < \mu < 1$ and $1 < \theta < \infty$. The terms c_x and c_y indicate the chronological orderings of different versions of a software program. The expression $\mu^{|c_x - c_y|}$ is a fine-grain measurement, because x and y have a small degree of difference.

In this function, the term $sim(x, y)$ is computed based on Eq. 5.1.1 and the main significant value comes from the expression $|c_x - c_y|$. When this value is large, it means that x and y are less compatible; otherwise, they are more compatible.

The numerical reasoning is about the similarity between two numeric values of a concept such as CPU speed or RAM size.

Definition 5.1.4. [57] Let a and b be numeric values and c a concept. The numerical reasoning function $Sim : \mathbb{R} \times \mathbb{R} \times C \rightarrow [0, 1]$ is defined as:

$$Sim(a, b, c) = 1 - |\frac{a - b}{Max_c - Min_c}|, \tag{5.1.3}$$

where Max_c and Min_c are the minimum and maximum values being available for c.

5.1.2 Semantic-Based Behavior Composition

Given the notion of ontology and reasoning, each action handled by an available behavior is considered as a concept. To have a matchmaking between the action requested by a target and those available in the system, two sets of actions are defined in the framework. One set, denoted by A_t, contains the requested actions, and the other set, denoted by A_s, includes all actions handled by the available behaviors in the system [11, 8]. Given such sets, the target behavior $\mathcal{B}_t$, available behaviors $\mathcal{B}_i$ $(1 \leq i \leq n)$, and the system $\mathcal{S}$ are as defined at the end of Sect. 2.1, but without the constraint that $A_t \subseteq \cup_i A_i$.

Example 5.1.1. Consider requirements expressed as a target behavior and depicted in Fig. 5.2. The requested resources belong to the set:

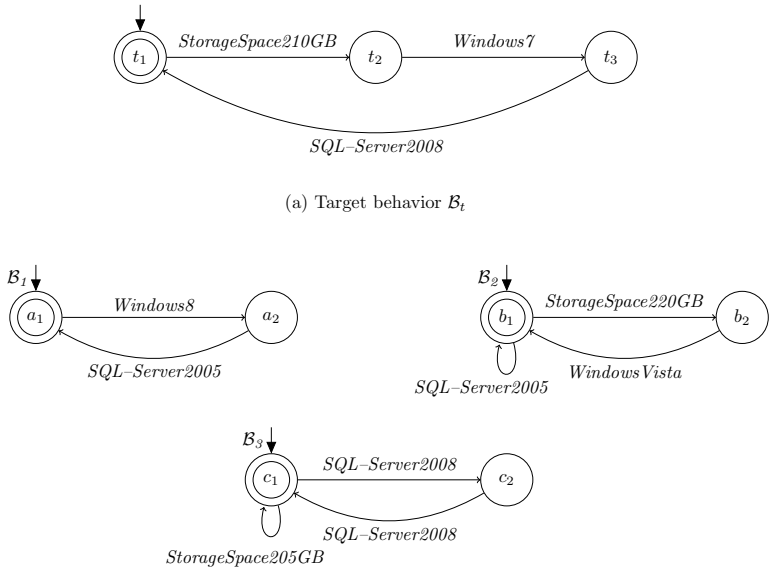
$$A_t = \{StorageSpace210GB, Windows7, SQL\text{–}Server2008\}.$$

Many of them are not available in the cloud. Besides, there are three available behaviors able to meet the target. For instance, behavior $\mathcal{B}_1$ is able to offer the resource set:

$$A_1 = \{Windows8, SQL\text{–}Server2005\}.$$

Though such resources do not have the same name as those of the requested resources, they can be similar or have the same functionality. In this end, a part of cloud ontology graph is illustrated in Fig. 5.3 to indicate the relations among resources. ■

Taking into account the three types of resource reasoning, the definition of largest ND-simulation relation must be revisited to support them. Let v_s and v_t be the numeric values for a_s and a_t, respectively, and c is a concept carried by both a_t and



(a) Target behavior $\mathcal{B}_t$

(b) Available behaviors $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$

Figure 5.2: A target behavior and available behaviors handling cloud resources

a_s. Moreover, assuming that τ_1, τ_2, and τ_3 be thresholds. An ND-simulation relation of $\mathcal{B}_t$ by $\mathcal{S}$ is a relation $R \subseteq B_t \times S$ such that $\langle t, s\rangle \in R$ implies:

Extension of Def. 2.2.1

1. if $t \in F_t$, then $s \in F$;
2. for all transitions $\langle t, a_t, t'\rangle \in \delta_t$:
   - there exists a transition $\langle s, a_s, k, s'\rangle \in \delta$;
   - for all transitions $\langle s, a_s, k, s'\rangle \in \delta$, $\langle t', s'\rangle \in R$ and:
     - $sim(a_t, a_s) \geq \tau_1$ if reasoning is similarity,
     - $compat(a_t, a_s) \geq \tau_2$ if reasoning is compatibility,
     - $Sim(v_s, v_t, c) \geq \tau_3$ if reasoning is numerical.

For each type of reasoning, a condition is provided to ensure that at each step of resource execution, the degree of match between a requested resource and an available

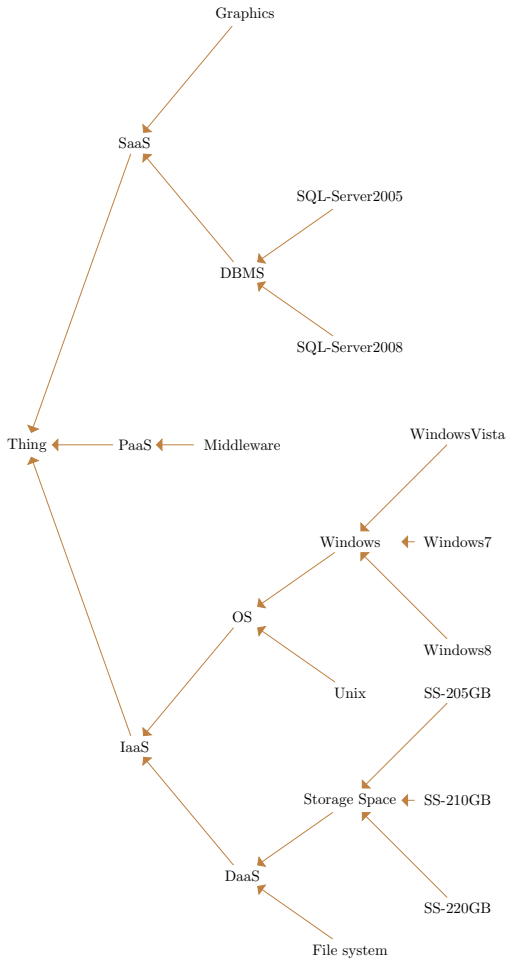


Figure 5.3: A part of an ontology for the cloud resources

resource is equal to or greater than a predefined threshold.

Based on the extension of the largest ND-simulation relation, the notion of controller generator also requires a revision to consider the reasoning conditions. Formally, the controller generator CG of $\mathcal{B}_t$ on $\mathcal{S}$ is $\langle \Sigma, A_t, A_s, I_n, \xi, \omega \rangle$, where:

Extension of Def. 2.2.2

1. $\Sigma = \{\langle t, s\rangle \in B_t \times S \mid t \preceq s\}$ is the set of CG states made by all pairs of $\mathcal{B}_t$ and $\mathcal{S}$ states that belong to the largest ND-simulation relation;
2. ξ is the transition relation, where $\sigma \xrightarrow{a_t,a_s,k} \sigma'$ in ξ, if and only if:
   - there is a transition $t \xrightarrow{a_t} t'$ in $\mathcal{B}_t$;
   - there is a transition $s \xrightarrow{a_s,k} s'$ in $\mathcal{S}$;
   - in the case of similarity reasoning, the condition $sim(a_t, a_s) \geq \tau_1$ holds;
   - in the case of compatibility reasoning, the condition $compat(a_t, a_s) \geq \tau_2$ holds;
   - in the case of numerical reasoning, the condition $Sim(v_s, v_t, c) \geq \tau_3$ holds;
   - for all $\langle t'', s''\rangle \in B_t \times S$, such that $s \xrightarrow{a_s,k} s''$ in $\mathcal{S}$ and $t \xrightarrow{a_t} t''$ in $\mathcal{B}_t$, it is the case that $\langle t'', s''\rangle \in \Sigma$;
3. $\omega : \Sigma \times A_t \times A_s \to 2^{I_n}$ is the output function with $\omega(\sigma, a_t, a_s) = \{k \mid \exists \sigma' \in \Sigma$ such that $\langle \sigma, \langle a_t, a_s, k\rangle, \sigma'\rangle \in \xi\}$.

Example 5.1.2. The largest ND-simulation relation for Example 5.1.1 is computed based on compatibility and numerical reasoning, which are done from the ontology graph in Fig. 5.3. More precisely, the compatibility reasoning is used for both *Windows* and *SQL–Server*, whereas a numerical reasoning is used for *StorageSpace*. Given the simulation relation:

$$R = \{\langle t_1, \langle a_1, b_1, c_1\rangle\rangle, \langle t_2, \langle a_1, b_2, c_1\rangle\rangle, \langle t_2, \langle a_1, b_1, c_1\rangle\rangle, \langle t_3, \langle a_1, b_1, c_1\rangle\rangle, \langle t_3, \langle a_2, b_1, c_1\rangle\rangle\},$$

the controller generator, illustrated in Fig. 5.4, is synthesized. All transitions of the controller generator are labeled by a pair of resources, namely a requested resource of the target and a similar resource to the request handled by an available behavior. For instance, due to the compatibility between *WindowsVista* and *Windows7*, the former can be offered to the target behavior.

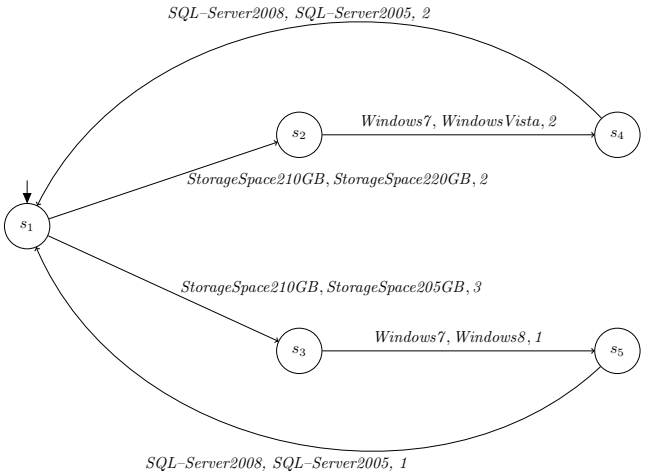


Figure 5.4: The controller generator

Given the possible generated controllers that can be extracted from the controller generator, the transition relations of two of them are:

$$\begin{aligned} P_1 \quad : \quad & \{\, \langle s_1, \mathit{StorageSpace210GB,StorageSpace220GB}, 2, s_2 \rangle, \\ & \langle s_2, \mathit{Windows7,WindowsVista}, 2, s_4 \rangle, \\ & \langle s_4, \mathit{SQL–Server2008,SQL–Server2005}, 2, s_1 \rangle \}. \end{aligned}$$

$$\begin{aligned} P_2 \quad : \quad & \{\, \langle s_1, \mathit{StorageSpace210GB,StorageSpace205GB}, 3, s_3 \rangle, \\ & \langle s_3, \mathit{Windows7,Windows8}, 1, s_5 \rangle, \\ & \langle s_5, \mathit{SQL–Server2008,SQL–Server2005}, 1, s_1 \rangle \}. \end{aligned}$$

It is supposed that the predefined threshold for compatibility and numerical reasoning are $\tau_2 = 0.7$ and $\tau_3 = 0.3$, respectively. Moreover, let ρ in Eq. 5.1.1 be set to 0.5, and μ and θ in Eq. 5.1.2 be set to 0.8 and 10, respectively. Given the following amounts, separately computed for each transition in the generated controllers, the average degree of match for the generated controllers P_1 and P_2 are 0.68 and 0.79, respectively. For instance, the average degree of match for P_1 is calculated as (0.33+0.88+0.83)/3.

$$Sim(205GB, 210GB, StorageSpace) = 0.66,$$
$$Sim(220GB, 210GB, StorageSpace) = 0.33,$$
$$compat(Windows7, Windows8) = compat(WindowsVista, Windows8) = 0.88,$$
$$sim(SQL\text{–}Server2008, SQL\text{–}Server2005) = 0.83,$$

where $|c_{Windows7} - c_{Windows8}| = 1$, $|c_{Windows7} - c_{WindowsVista}| = 1$, $|\alpha(Windows8)| = 5$, and $|\alpha(SQL\text{–}Server2008) \cap \alpha(SQL\text{–}Server2005)| = 3$. ■

5.1.3 Implementation of Resource Reasoning in SMV/TLV

For providing a matchmaking between actions of a target behavior and those of available behaviors, first of all, for each type of resource reasoning, a variable, named `threshold`, is declared in the system module of SMV. Through this variable, the target behavior is able to assign its acceptable threshold for the degree of match between its requested action and the one in system $\mathcal{S}$. The type of this variable must be integer, since the real numbers are not supported in SMV. For instance, a real number 0.7 is considered as an integer number 7. Given the ranges of resource reasoning functions, the domain of `threshold` for both similarity and numerical reasonings is the integer numbers in the interval [0,10] and for compatibility reasoning is the integer number in the interval (0,20). Such domains are appropriate if only one digit of decimal precision is taken into account for the thresholds in the extension of Def. 2.2.1. Note that, the values returned from the reasoning functions defined from Def. 5.1.2 to 5.1.4 cannot be computed by SMV/TLV.

The variable `threshold` is considered as an argument in both target module and available behavior modules. In the former, it is regarded as an output argument in which at each step of an action request, its threshold, denoting the acceptable degree of match between the action and the ones in available behaviors, is released. In contrast, this variable is an input argument in the latter through which the precomputed degree of match between a requested action and similar ones in the current state of a behavior is compared with the released threshold. Note that, in the transition part of the available behavior module, the condition for the threshold is associated with

```
MODULE Target(resource,threshold)
 VAR
   state : {start_st,t1,t2,t3};
 INIT
   state = start_st & resource = start_op & threshold = 0
 TRANS
  case
    state = start_st & resource = start_op & threshold = 0 : next(state) = t1 &
          next(resource) in {StorageSpace210GB} & next(threshold) = 3;
    state = t1 & resource in {StorageSpace210GB} & threshold = 3 : next(state) = t2 &
          next(resource) in {Windows7} & next(threshold) = 8;
    state = t2 & resource in {Windows7} & threshold = 8 : next(state) = t3 &
          next(resource) in {SQL-Server2008} & next(threshold) = 7;
    state = t3 & in {SQL-Server2008} & threshold = 7 : next(state) = t1 &
          next(resource) in {StorageSpace210GB} & next(threshold) = 3;
   esac
  DEFINE
   initial := state = start_st & resource = start_op & threshold = 0;
   final := state in {t1};


MODULE Behavior1(index,resource,threshold)
 VAR
   state : {start_st,a1,a2};
 INIT
   state = start_st
 TRANS
  case
   state = start_st & resource = start_op & threshold = 0 & index = 0 : next(state) in {a1};
   (index != 1) : next(state) = state;
   (state=a1 & resource in {Windows8,Windows7} & threshold<=8) : next(state) in {a2};
   (state=a2 & resource in {SQL-Server2005,SQL-Server2008} & threshold<=8) : next(state) in {a1};
  esac
 DEFINE
  initial := state = start_st & resource = start_op & threshold = 0 & index = 0;
  failure := index = 1 & !((state = a1 & resource in {Windows8,Windows7} & threshold<=8) |
             (state = a2 & resource in {SQL-Server2005,SQL-Server2008} & threshold<=8));
  final := state in {a1};
```

Figure 5.5: The modules of the target behavior and an available behavior in SMV

an action handled by the behavior and the one requested by the target. Hence, in each transition of the module, a set is provided to encompass both actions.

Example 5.1.3. Figure 5.5 illustrates the SMV modules of the target $\mathcal{B}_t$ and behavior $\mathcal{B}_1$ in Example 5.1.1. Given the transitions in the target behavior module, it is supposed that:

- the threshold τ_1 for the degree of match between *SQL–Server2005* and the similar resource in the system is 7,
- the threshold τ_2 for the degree of match between *Windows7* and the similar resource in the system is 8,
- the threshold τ_3 for the degree of match between *StorageSpace210GB* and the similar resource in the system is 3.

For simplification, although three different variables should be declared for τ_1, τ_2 and τ_3 in the implementation of this example, only one variable `threshold` is declared.

Given the transition part of the target behavior module, this variable is assigned with three values of 3, 8, and 7 to consider the acceptable thresholds for numerical, compatibility, and similarity reasonings, respectively. These values are inputs in the module of available behaviors. For instance, the requested resource *SQL–Server2008* can be matched with the resource *SQL–Server2005* in the module `behavior1`, since the current input of threshold in this module has the value of 7, and the degree of similarity between the resources is 8 ($sim(SQL\text{–}Server2008, SQL\text{–}Server2005) = 0.83$).

Appendix E.1 provides the details about all the SMV modules of Example 5.1.1 along with the TLV output.

5.1.4 Experimental Results

Through the implementation of resource reasoning in SMV, an experiment has been provided. Given a fixed number of target behaviors, the experiment shows the effects of similarity reasoning on the number of realized target behaviors.

The Effect of Similarity Reasoning on the Realized Target Behaviors

The assumption is that there is one available behavior, carrying 16 different actions. Besides, it is supposed that the number of target behaviors varies from 2 to 12. Each target behavior requests four different actions such that from those, one action is randomly chosen to be possibly matched with a similar one in the available behavior. The predefined threshold in the target behavior can be 2, 4, 6, or 8. Likewise, in the available behavior, the degree of match between an action and the one requested by the target behavior is randomly selected between 1 and 10.

Given the graphs illustrated in Fig. 5.6, the horizontal axes represent the number of target behaviors, and the vertical axes indicate the average number of realized target behaviors, where for each datum (point in a curve), it is calculated after 10 execution of TLV program. To investigate the average number of realized target behaviors in the larger scale of actions, the graphs depicted in Fig. 5.6-(b) and Fig. 5.6-(c) have been provided. In the former, the available behavior carries 32 different actions, and in the latter, the number of actions is extended to 48. Notably that, in the both

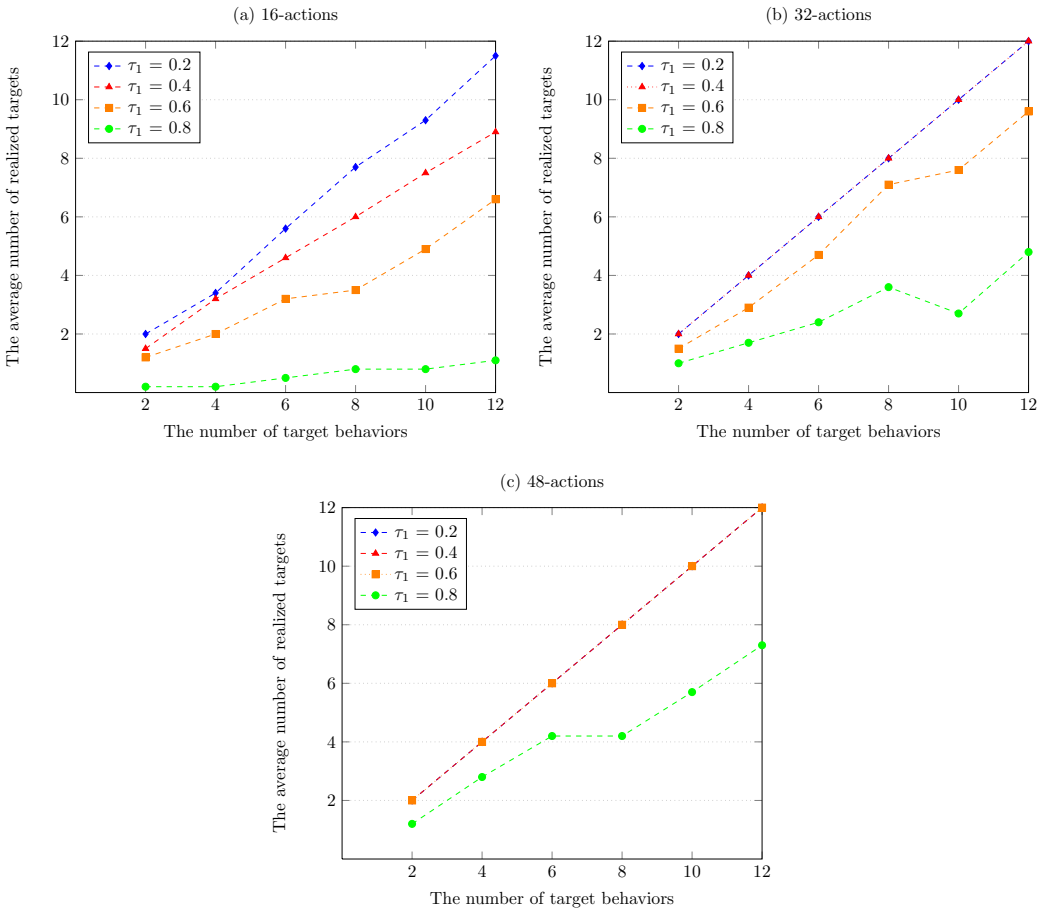


Figure 5.6: The relationships between the realized target behaviors and similarity reasoning under different scale of actions

graphs, the scenario for the requested actions of the target behaviors is similar to the one assumed for Fig. 5.6-(a).

It can be comprehended from the graphs that when the rate of the threshold increases, the average number of realized target behaviors decreases sharply. Furthermore, the growth in the number of actions leads to the increase in the average number of realized target behaviors. Besides, a comparison between the curves and the rates of threshold indicates that the curves, having the lower rates of threshold, are closer to each other, and such curves are mapped onto each other when the number of actions increases.

5.2 Reasoning Based on Preferences

The techniques based on explicit qualitative preferences sit at the opposite end of the reasoning and semantic-based techniques. Regardless of the precision and computational complexity inherent to the reasoning metrics, they allow for expressing preferences about artifacts more naturally and directly. On the one hand, they give strong control to clients or desired targets in regard to specific needs. On the other hand, they help alleviate the number of undesirable matches. They are particularly convenient when there is not enough semantic information about artifacts to calculate the degree of match.

Assuming that resources have a set of attributes $Att = \{att_1, \ldots, att_l\}$. Each attribute att_i can be associated with a domain of values $V_i = \text{dom}(att_i)$. A valuation for a set of attributes Att is a function $\nu : Att \rightarrow V_1 \cup \cdots \cup V_l$, assigning a value $\nu(att_i) \in V_i$ to every attribute $att_i \in Att$. The term $\text{val}(Att)$ denotes the set of all valuations over Att. The notion of valuation entails the idea of resource variability due to dynamic changes of attribute values. Given a set of attributes Att, a preference can be expressed about an attribute $att \in Att$.

Definition 5.2.1. [30] A preference $\mathcal{P} = (att, <^{\mathcal{P}})$ is a strict partial order, where $<^{\mathcal{P}} \subseteq \text{dom}(att) \times \text{dom}(att)$. If $x, y \in \text{dom}(att)$, then $x <^{\mathcal{P}} y$ is expressed as "y is preferred rather than x".

Based on the Def. 5.2.1, the intended preference model introduced hereafter has been already defined to support complex database queries to match user preferences closely [38]. The principle behind a preference-based database search engine is query relaxation. Its adoption leads to reasoning about approximate-match query results, which is more appropriate, for instance, in real-world big-data applications. Later, a semantic ontology of user preferences, which largely rests on this work, has been proposed in the context of web-service discovery and ranking [30]. Basically, the authors follow the same mathematical formulation for preferences. Nevertheless, depending on the context in which the model is used, different interpretations can be adopted when the condition associated with a given preference is not satisfied. Given a set of available resources with their own attributes, the realization of a request, through the synthesis of a decision maker, from functional and nonfunctional requirements

expressed in terms of preferences among attribute values, is, in some sense, partly reminiscent of a complex preference query.

The preference model distinguishes between atomic and composite preferences. The former are subdivided into qualitative and quantitative preferences. The preferences expressed over a single attribute should not be contradictory.

5.2.1 Qualitative Atomic Preferences

A natural approach to specify qualitative preferences is to use sets of lexical terms instead of rating values. Given an attribute $att \in Att$, the disjoint sets *Fav*, *Alt*, and *Dis* included in dom(att) represent the favorite, alternative, and disliked values for attribute att, respectively. Let $x, y \in \text{dom}(att)$, the following definitions are proposed for qualitative preferences.

Definition 5.2.2. $\mathcal{P} = \boldsymbol{Fav}$ is a favorite preference, if:

$$x <^{\boldsymbol{Fav}} y \quad \text{iff} \quad y \in Fav \wedge x \notin Fav. \tag{5.2.1}$$

In this kind of preference, the resource with the attribute att having a value that does not belong to *Fav* is irrelevant.

Definition 5.2.3. $\mathcal{P} = \boldsymbol{Dis}$ is a dislike preference, if:

$$x <^{\boldsymbol{Dis}} y \quad \text{iff} \quad y \notin Dis \wedge x \in Dis. \tag{5.2.2}$$

The dislike preference is the opposite of favorite one. A resource with the attribute att is irrelevant when the attribute value belongs to *Dis*.

Definition 5.2.4. $\mathcal{P} = \boldsymbol{Fav/Alt}$ is a favorite/alternative preference, if:

$$\begin{aligned} x <^{\boldsymbol{Fav/Alt}} y \quad \text{iff} \quad & (y \in Fav \wedge x \in Alt) \vee \\ & (y \in Fav \cup Alt \wedge x \notin Fav \cup Alt). \end{aligned} \tag{5.2.3}$$

Such sort of preference gives an advantage to a resource with an attribute having a value that belongs to *Fav*, if they exist, without ignoring those for which the value belongs to *Alt*. The resource for which the value of the attribute does not belong to *Fav* or *Alt* is irrelevant.

Definition 5.2.5. $\mathcal{P} = \boldsymbol{Fav/Dis}$ is a favorite/dislike preference, if:

$$x <^{\boldsymbol{Fav/Dis}} y \quad \text{iff} \quad (y \in Fav \wedge x \notin Fav) \vee (y \notin Fav \cup Dis \wedge x \in Dis). \tag{5.2.4}$$

Given a favorite/dislike preference, a resource should preferably have a value in *Fav* for the corresponding attribute. Otherwise, the value should not belong to *Dis*. The condition to ignore a resource is the same as for the dislike preference. Notice that, the authors of [30, 38] give two different formulations of favorite/dislike. Both include an inconsistency. The provided definition corrects these mistakes.

5.2.2 Quantitative Atomic Preferences

Many preferences are often expressed by using numerical values rather than lexical terms. Typically, the attribute domain is $\mathbb{N}$, $\mathbb{Z}$, or $\mathbb{R}$ (equipped with a subtractive operation). Given an attribute $att \in Att$, typical preferences over $\text{dom}(att)$ are lowest, highest, around, and score.

Definition 5.2.6. $\mathcal{P} = Lowest$ is a lowest preference, if:

$$x <^{Lowest} y \quad \text{iff} \quad x > y. \tag{5.2.5}$$

It is readily interpreted that the preference choices the resource, which its attribute has a lower value.

Definition 5.2.7. $\mathcal{P} = Highest$ is a highest preference, if:

$$x <^{Highest} y \quad \text{iff} \quad x < y. \tag{5.2.6}$$

This kind of preference is opposite to the lowest one.

Definition 5.2.8. Let $v \in \text{dom}(att)$, $\mathcal{P} = Around(v)$ is an around preference, if:

$$x <^{Around(v)} y \quad \text{iff} \quad |x - v| > |y - v|. \tag{5.2.7}$$

Given the distance of each value to a reference value identified as the operand of the preference, the closer value is determined through an around preference.

Definition 5.2.9. Let $f : \mathrm{dom}(att) \to \mathbb{R}$, $\mathcal{P} = \mathit{Score(f)}$ is a score preference, if:

$$x <^{Score(f)} y \quad \text{iff} \quad f(x) < f(y). \tag{5.2.8}$$

This preference uses a scoring function, taking an attribute value as its argument and returns a real value. Regarding two attribute values x and y, if the scoring function returns a higher value for y, it means that y is more preferred than x.

5.2.3 Composite Preferences

Generally, an action has more than one attribute. Preferences must then be composed to relate two or more preferences. Constructors for balanced (or Pareto), prioritized, and numerical preferences have been introduced for that purpose.

Definition 5.2.10. Let $\mathcal{P}_1 = (att_1, <^{\mathcal{P}_1})$ and $\mathcal{P}_2 = (att_2, <^{\mathcal{P}_2})$ be two preferences, and $x = (x_1, x_2), y = (y_1, y_2) \in \mathrm{dom}(att_1) \times \mathrm{dom}(att_2)$. $\mathit{Balance} = (att_1 \cup att_2, <^{Balance})$ is a balanced preference, if:

$$\begin{aligned} x <^{Balance} y \quad \text{iff} \quad & (x_1 <^{\mathcal{P}_1} y_1 \wedge (x_2 <^{\mathcal{P}_2} y_2 \vee x_2 = y_2)) \vee \\ & (x_2 <^{\mathcal{P}_2} y_2 \wedge (x_1 <^{\mathcal{P}_1} y_1 \vee x_1 = y_1)). \end{aligned} \tag{5.2.9}$$

This preference is a combination of two preference $\mathcal{P}_1$ and $\mathcal{P}_2$, which uses Pareto optimality principle. In a sense, both preferences have the same importance.

Definition 5.2.11. Given two preferences $\mathcal{P}_1$ and $\mathcal{P}_2$ and two different domains, $\mathit{Priority} = (att_1 \cup att_2, <^{Priority})$ is a prioritized preference, if:

$$x <^{Priority} y \quad \text{iff} \quad (x_1 <^{\mathcal{P}_1} y_1 \vee (x_1 = y_1 \wedge x_2 <^{\mathcal{P}_2} y_2). \tag{5.2.10}$$

In the prioritized preference, combining two preference $\mathcal{P}_1$ and $\mathcal{P}_2$, the former preference is more important than the latter. The evaluation of $\mathcal{P}_2$ must be done, only if x_1 and y_1 have the same priority.

Definition 5.2.12. Let $f_1 : \mathrm{dom}(att_1) \to \mathbb{N}$ and $f_2 : \mathrm{dom}(att_2) \to \mathbb{N}$ be two functions that transform numerical values or lexical terms of two different domains into natural

numbers. Let $F : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$, then the following strict order is a numerical preference:

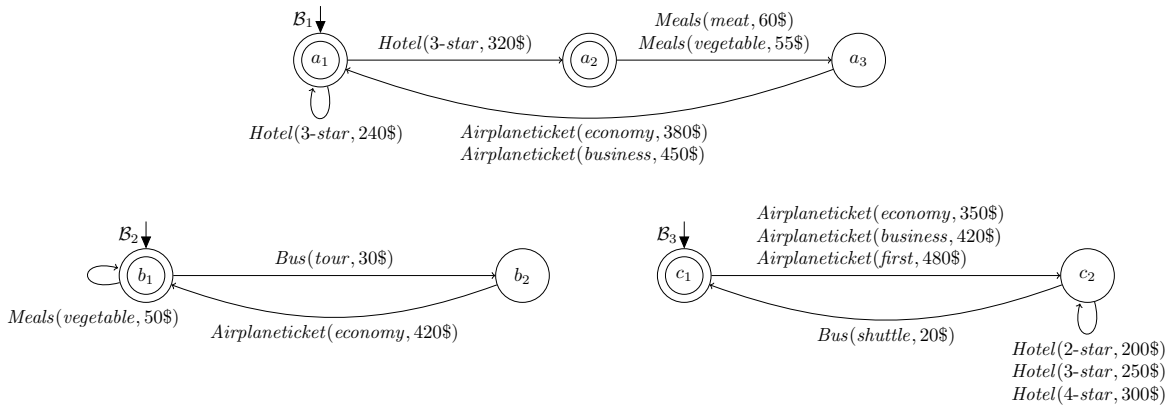
$$x <^{Numerical} y \quad \text{iff} \quad F(f_1(x_1), f_2(x_2)) < F(f_1(y_1), f_2(y_2)). \tag{5.2.11}$$

Numerical preference combines a set of score preferences, and generates a real number giving information about the composite or global preference through a function which its arguments are values returned by the score preferences. For instance, let $F : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$, a weighted function $F(n_1, n_2) = 0.3n_1 + 0.7n_2$ can be considered as a weighted preference, which is a particular kind of numerical preference.

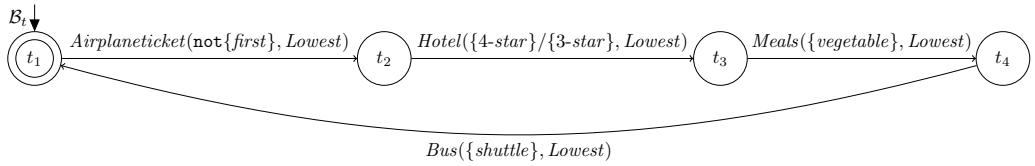
5.2.4 Preference-Based Behavior Composition

The formal representation for behaviors need to be changed to take into account attributes and preferences. Each behavior $\mathcal{B}_i$ is represented by a finite-state transition system $\langle B_i, A_i, \alpha_i, \delta_i, b_{i0}, F_i \rangle$, where $\alpha_i : A_i \to 2^{Att}$ is a function assigning a subset of attributes of Att to every action. The same actions of different behaviors have the same attributes (i.e., $\alpha_i(a) = \alpha_j(a)$, if $a \in A_i \cap A_j$). Likewise, the system is $\mathcal{S} = \langle S, A_s, I_n, \alpha, \delta, s_0, F \rangle$, where $S = B_1 \times \cdots \times B_n$, with $s_0 = \langle b_{10}, \ldots, b_{n0} \rangle$ and $F = F_1 \times \cdots \times F_n$; $A_s = \cup_i A_i$ is the set of actions; $\alpha_i(a) = \alpha(a)$ for $a \in A_i$; and $\delta \subseteq S \times A_s \times I_n \times S$ is the transition relation defined in the usual way, except that each transition is labeled by an action and an index that belongs to I_n, which indicates the behavior that may perform the action. The target behavior is $\mathcal{B}_t = \langle B_t, A_t, \alpha_t, \delta_t, b_{t0}, F_t \rangle$, where $A_t \subseteq A_s$. For any $a \in A_t$, $\alpha_t(a) \subseteq \alpha(a)$, namely an action of the target behavior has the same attributes as the ones of the corresponding action in the system.

Example 5.2.1. Figure 5.7 shows the three available behaviors on the web which offer traveling services, namely $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$. The goal is to realize an on-line travel agency with the target behavior $\mathcal{B}_t$. A transition labeled with more than one service (action) is just a more compact notation to represent multiple transitions. All the services of behaviors have two attributes and their values are between parenthesis. The value of the second attribute is the price a traveler will pay for the corresponding service. The preferences for the corresponding services appear in the target behavior. The dislike preference "not{*first*}" indicates that the customer does not want a first



(a) Available behaviors $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$



(b) Target behavior $\mathcal{B}_t$

Figure 5.7: The available behaviors and a target behavior for an on-line travel agency

class flight. He also prefers a 4-star hotel, but a 3-star hotel is acceptable. This is a favorite/alternative preference. The preferences for meals and bus are favorite preferences. The preference for the second attribute is a quantitative preference. It indicates that the customer always looks for the lowest prices. ■

Values must be assigned to attributes via valuations. To provide a flexible model, the way the values are assigned to attributes depends on transitions, that is, each transition is labeled by an instance of an action (an action with specific values for its attributes). In this sense, a valuation ν is a conditional valuation and $\nu(att|\langle s, a, k, s'\rangle)$ (or shortly $\nu(att)$ when the context is clear), with $\langle s, a, k, s'\rangle \in \delta$ and $att \in \alpha(a)$, denotes the value of att. The counterpart of ν for $\mathcal{B}_t$ is $\mathcal{P}(att)$, but its range is atomic preferences (*Indifference* is the default preference). For instance, $\mathcal{P}(att) = \boldsymbol{Fav/Alt}$ for a favorite/alternative preference with the sets *Fav* and *Alt* as arguments.

The main notion of ND-simulation relation has been adapted, so that the match between actions is not strict, but depends on preferences. A filter is applied on the actions with respect to Eqs. 5.2.1 to 5.2.4. An ND-simulation relation of $\mathcal{B}_t$ by $\mathcal{S}$ is a relation $R \subseteq B_t \times S$ such that $\langle t, s \rangle \in R$ implies:

Extension of Def. 2.2.1

1. if $t \in F_t$, then $s \in F$;
2. for all transitions $\langle t, a, t' \rangle \in \delta_t$:
   - there exists a transition $\langle s, a, k, s' \rangle \in \delta$;
   - for all transitions $\langle s, a, k, s' \rangle \in \delta$, $\langle t', s' \rangle \in R$ and for all $att \in \alpha_t(a)$:
     - $\nu(att) \in Fav$ if $\mathcal{P}(att) = \boldsymbol{Fav}$,
     - $\nu(att) \notin Dis$ if $\mathcal{P}(att) = \boldsymbol{Dis}$,
     - $\nu(att) \in Fav \cup Alt$ if $\mathcal{P}(att) = \boldsymbol{Fav/Alt}$,
     - $\nu(att) \notin Dis$ if $\mathcal{P}(att) = \boldsymbol{Fav/Dis}$.

This version of the ND-simulation relation can be easily implemented through a fixpoint calculation procedure to obtain the largest relation R. A final decision (acceptance or rejection of an action offered by an available behavior) cannot be taken due to the fact that the transitions are examined one at a time with local information. The characterization of the "best" compositions (generated controllers) relies, among other things, on the composite preferences used to rank all candidate compositions.

Notice that, the conditions related to the quantitative preferences are not considered in the formulation of the largest ND-simulation relation, and they are only taken into account when the compositions should be ranked or evaluated after the synthesis procedure.

Given the extension in the largest ND-simulation relation, the notion of controller generator also requires a revision to consider the qualitative preferences. Formally, the controller generator CG of $\mathcal{B}_t$ on $\mathcal{S}$ is $\langle \Sigma, A_t, I_n, \alpha_t, \xi, \omega \rangle$, where:

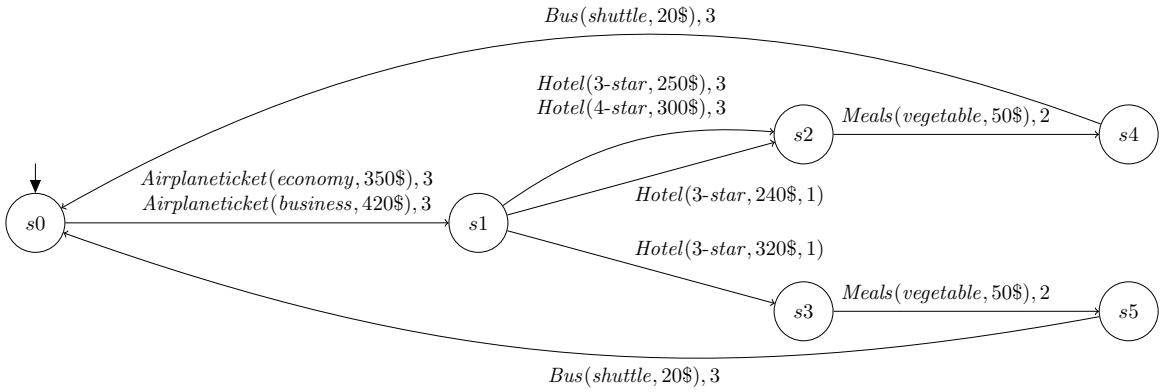


Figure 5.8: The controller generator of the travel agency system

Extension of Def. 2.2.2

1. $\Sigma = \{\langle t, s\rangle \in B_t \times S \mid t \preceq s\}$ is the set of CG states made by all pairs of $\mathcal{B}_t$ and $\mathcal{S}$ states that belong to the largest ND-simulation relation;
2. ξ is the transition relation, where $\sigma \xrightarrow{a,k} \sigma'$ in ξ, if and only if:
   - there is a transition $t \xrightarrow{a} t'$ in $\mathcal{B}_t$;
   - there is a transition $s \xrightarrow{a,k} s'$ in $\mathcal{S}$;
   - for all $att \in \alpha_t(a)$:
     (a) if $\mathcal{P}(att) = \boldsymbol{Fav}$, then $\nu(att) \in Fav$ holds;
     (b) if $\mathcal{P}(att) = \boldsymbol{Dis}$, then $\nu(att) \notin Dis$ holds;
     (c) if $\mathcal{P}(att) = \boldsymbol{Fav/Alt}$, then $\nu(att) \in Fav \cup Alt$ holds;
     (d) if $\mathcal{P}(att) = \boldsymbol{Fav/Dis}$, then $\nu(att) \notin Dis$ holds;
   - for all $\langle t'', s''\rangle \in B_t \times S$, such that $s \xrightarrow{a,k} s''$ in $\mathcal{S}$ and $t \xrightarrow{a} t''$ in $\mathcal{B}_t$, it is the case that $\langle t'', s''\rangle \in \Sigma$;
3. $\omega : \Sigma \times A_t \to 2^{I_n}$ is the output function with $\omega(\sigma, a) = \{k \mid \exists \sigma' \in \Sigma$ such that $\langle \sigma, \langle a, k\rangle, \sigma'\rangle \in \xi\}$.

Example 5.2.2. Figure 5.8 shows the controller generator synthesized from $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$, and $\mathcal{B}_t$ introduced in Fig. 5.7. Six generated controllers can be identified form

the controller generator. Four are deterministic:

$$
\begin{aligned}
P_1 \;:\; \{\, & \langle s_0, \mathit{Airplaneticket}(\mathit{economy}, 350\$), 3, s_1\rangle,\\
& \langle s_1, \mathit{Hotel}(4\text{-}\mathit{star}, 300\$), 3, s_2\rangle,\\
& \langle s_2, \mathit{Meals}(\mathit{vegetable}, 50\$), 2, s_4\rangle,\\
& \langle s_4, \mathit{Bus}(\mathit{shuttle}, 20\$), 3, s_0\rangle\}.
\end{aligned}
$$

$$
\begin{aligned}
P_2 \;:\; \{\, & \langle s_0, \mathit{Airplaneticket}(\mathit{economy}, 350\$), 3, s_1\rangle,\\
& \langle s_1, \mathit{Hotel}(3\text{-}\mathit{star}, 250\$), 3, s_2\rangle,\\
& \langle s_2, \mathit{Meals}(\mathit{vegetable}, 50\$), 2, s_4\rangle,\\
& \langle s_4, \mathit{Bus}(\mathit{shuttle}, 20\$), 3, s_0\rangle\}.
\end{aligned}
$$

$$
\begin{aligned}
P_3 \;:\; \{\, & \langle s_0, \mathit{Airplaneticket}(\mathit{business}, 420\$), 3, s_1\rangle,\\
& \langle s_1, \mathit{Hotel}(3\text{-}\mathit{star}, 250\$), 3, s_2\rangle,\\
& \langle s_2, \mathit{Meals}(\mathit{vegetable}, 50\$), 2, s_4\rangle,\\
& \langle s_4, \mathit{Bus}(\mathit{shuttle}, 20\$), 3, s_0\rangle\}.
\end{aligned}
$$

$$
\begin{aligned}
P_4 \;:\; \{\, & \langle s_0, \mathit{Airplaneticket}(\mathit{business}, 420\$), 3, s_1\rangle,\\
& \langle s_1, \mathit{Hotel}(4\text{-}\mathit{star}, 300\$), 3, s_2\rangle,\\
& \langle s_2, \mathit{Meals}(\mathit{vegetable}, 50\$), 2, s_4\rangle,\\
& \langle s_4, \mathit{Bus}(\mathit{shuttle}, 20\$), 3, s_0\rangle\}.
\end{aligned}
$$

Two are nondeterministic:

$$
\begin{aligned}
P_5 \;:\; \{\, & \langle s_0, \mathit{Airplaneticket}(\mathit{business}, 420\$), 3, s_1\rangle, \\
& \langle s_1, \mathit{Hotel}(3\text{-}\mathit{star}, 240\$), 1, s_2\rangle, \langle s_1, \mathit{Hotel}(3\text{-}\mathit{star}, 320\$), 1, s_3\rangle, \\
& \langle s_2, \mathit{Meals}(\mathit{vegetable}, 50\$), 2, s_4\rangle, \langle s_3, \mathit{Meals}(\mathit{vegetable}, 50\$), 2, s_5\rangle, \\
& \langle s_4, \mathit{Bus}(\mathit{shuttle}, 20\$), 3, s_0\rangle, \langle s_5, \mathit{Bus}(\mathit{shuttle}, 20\$), 3, s_0\rangle\}.
\end{aligned}
$$

$$
\begin{aligned}
P_6 \;:\; \{\, & \langle s_0, \mathit{Airplaneticket}(\mathit{economy}, 350\$), 3, s_1\rangle, \\
& \langle s_1, \mathit{Hotel}(3\text{-}\mathit{star}, 240\$), 1, s_2\rangle, \langle s_1, \mathit{Hotel}(3\text{-}\mathit{star}, 320\$), 1, s_3\rangle, \\
& \langle s_2, \mathit{Meals}(\mathit{vegetable}, 50\$), 2, s_4\rangle, \langle s_3, \mathit{Meals}(\mathit{vegetable}, 50\$), 2, s_5\rangle, \\
& \langle s_4, \mathit{Bus}(\mathit{shuttle}, 20\$), 3, s_0\rangle, \langle s_5, \mathit{Bus}(\mathit{shuttle}, 20\$), 3, s_0\rangle\}.
\end{aligned}
$$

■

5.2.5 Implementation of Qualitative Preference in SMV/TLV

The original SMV module skeletons for a target behavior and available behaviors only support functional requirements (actions). They should be modified to include nonfunctional requirements (preferences and attributes) by exploiting set operators. The possible operators in SMV are inclusion, exclusion, and union denoted by `in`, `notin`, and `union`, respectively.

Example 5.2.3. The implemented modules of the target $\mathcal{B}_t$ and behavior $\mathcal{B}_1$ in Example 5.2.1 are demonstrated in Fig 5.9. In the target module, besides of a service, the desirable attribute values of the service is released through the argument `preference` defined in the module. Moreover, in this module, the sets related to the qualitative preferences are locally defined. For instance, the set `Alt_hotel` contains an alternative choice for the target with respect to the hotel reservation service. In the transition part of the target, the preference models are implemented through the operator set. As an example, the code `preference in (Fav_hotel union Alt_hotel)` denotes a favorite/alternative preference for the hotel reservation.

In the behavior module, a service along with the set of its attribute values is defined in each transition part of the module. The preferences generated from the target module are compared with the attribute values of services through the input

```
MODULE Target(service,preference)
 VAR
  state : {start_st,t1,t2,t3,t4};
  Fav_hotel : {star4};
  Alt_hotel : {star3};
  Dis_air : {first};
  Fav_meals : {vegetable};
  Fav_bus : {shuttle};
 INIT
  state = start_st & service = start_se & preference=start_pr
 TRANS
  case
   state = start_st & service = start_se & preference=start_pr: next(state) = t1 &
           next(service) in {Airplaneticket} & ((next(preference) in {economy,business,first})
           & (next(preference) notin Dis_air));
   state = t1 & service = Airplaneticket & ((preference in {economy,business,first}) &
           (preference notin Dis_air)) : next(state) in {t2} & next(service) in {Hotel}
           & next(preference) in (Fav_hotel union Alt_hotel);
   state = t2 & service = Hotel & preference in (Fav_hotel union Alt_hotel): next(state) = t3
           & next(service) in {Meals} & next(preference) in Fav_meals;
   state = t3 & service = Meals &  preference in Fav_meals : next(state) = t4 & next(service)
           in {Bus} & next(preference) in Fav_bus;
   state = t4 & service = Bus &  preference in Fav_bus : next(state) = t1 & next(service)
           in {Airplaneticket} & ((next(preference) in {economy,business,first}) &
           (next(preference) notin Dis_air));
  esac
 DEFINE
  initial := state=start_st & service=start_se & preference=start_pr;
  final := state in {t1}; -- final state(s)

MODULE Behavior1(index,service,Attr_values)
 VAR
  state : {start_st,a1,a2,a3};
 INIT
  state=start_st
 TRANS
  case
   state=start_st & service=start_se & Attr_values=start_pr & index=0: next(state)=a1;
   (index != 1) : next(state) = state;
   state=a1 & (service in {Hotel} & Attr_values in {star3}) : next(state) in {a1,a2};
   state=a2 & (service in {Meals} &  Attr_values in {vegetable,meat}): next(state) in {a3};
   state=a3 & (service in {Airplaneticket} & Attr_values in {economy,bussiness}): next(state) in {a1};
  esac
 DEFINE
  initial := state=start_st & service=start_se & index = 0 & Attr_values=start_pr;
  failure := index = 1 & !((state = a1 & (service in {Hotel} & Attr_values in {star3}))|
             (state = a2 & (service in {Meals} & Attr_values in {vegetable,meat}))|(state = a3
             & (service in {Airplaneticket} & Attr_values in {economy,business})));
  final := state in {a1,a2};
```

Figure 5.9: The SMV modules of the target behavior and an available behavior

argument (`Attr-values`) defined in the behavior module. For instance, since the behavior $\mathcal{B}_1$ contains a `3-star` in its attribute set associated with the hotel and the condition of `Attr_values in {3-star}` is satisfied, the service can be matched as an alternative choice for the target.

The details about all the SMV codes of Example 5.2.1, together with its TLV output are provided in Appendix E.2.

5.3 Reasoning Based on Semantics and Preferences

The notion of resource reasoning can be integrated in the preference based behavior composition framework to not only match the similar actions and attributes, but also match the similar attribute values and preferences. Furthermore, among a set of similar attribute values, choosing the one having the most degree of match requires an extension in the preference models to support similarity, compatibility, and numerical reasoning.

5.3.1 Integration of Resource Reasoning in Preference Model

An ontology graph can be provided to semantically define the attribute values of resources as concepts. Moreover, given the reasoning functions and such a graph, the degree of match between an attribute value and a preference can be calculated. Given such calculations and a preference, a new type of preference models can be defined over the attribute values to select the one that has more degree of match with the preference. Such preference models are: *similarity reasoning preference*, *compatibility reasoning preference*, and *numerical reasoning preference*, where their names are referred to the reasoning function used in them.

Given the models defined in Sect. 5.2, such type of preferences can be considered as the score preference defined in the quantitative atomic models. More precisely, the score in here is the degree of match between a preference and an attribute value that returned from a reasoning function.

Definition 5.3.1. Let $z \in \text{dom}(att)$ be a reference attribute value, $\mathcal{P} = \textit{Similar(z)}$ is a similarity reasoning preference, if:

$$x <^{\textit{Similar(z)}} y \quad \text{iff} \quad sim(z, x) < sim(z, y). \tag{5.3.1}$$

Given the two attribute values x and y, if the similarity function, defined in Eq. 5.1.1, returns a higher value for y with respect to z than x, it means that y is more preferred than x.

Definition 5.3.2. Let z be a reference attribute value, $\mathcal{P} = \textit{Compatible(z)}$ is a

compatibility reasoning preference, if:

$$x <^{Compatible(z)} y \quad \text{iff} \quad compat(z,x) < compat(z,y). \tag{5.3.2}$$

Having considered the attribute values x and y, if the compatibility reasoning function in Eq. 5.1.2 returns a higher value for y with respect to z than x, it means that y is more preferred than x.

The preference models defined for similarity and compatibility reasoning can be considered as particular cases of score preferences through which the scoring function, having two attribute values as its arguments, returns a real value to indicate the degree of similarity or compatibility between them. As mentioned in Sect. 5.1.1, the returned score or real value for similarity and compatibility reasoning preferences can be in the intervals of [0,1] and (0,2), respectively.

Definition 5.3.3. Let z be a reference attribute value, $\mathcal{P} = Numeric(z,att)$ is a numerical reasoning preference, if:

$$x <^{Numeric(z,att)} y \quad \text{iff} \quad Sim(x,z,att) < Sim(y,z,att). \tag{5.3.3}$$

Given the attribute values x and y, if the numerical reasoning function in Eq. 5.1.3 returns a higher value for y with respect to z than x, it means that y is closer to z rather than x.

5.3.2 A Hybrid Behavior Composition Framework

Consider two alternative resources. A user may only desire the resource satisfying his preference. Whereas, for another user, the two resources may be indifferent from his point of view. In such a case, the symmetric part of a user preference is considered to reflect the similarity or indifference between resources [17]. A hybrid framework that takes into account both similarity and preference can support the two different user' views.

The largest ND-simulation relation proposed for preference-based behavior composition (see Sect. 5.2.4) requires also a revision to incorporate resource reasoning. Let v_{s0}, v_{t0}, v_{s1}, v_{t1}, v_{s2}, and v_{t2} be the numeric values for a_s, a_t, att_s, att_t, $x_s \in \nu(att_s)$

and $x_t \in \mathcal{P}(att_t)$, respectively, and c_0 be the common concept carried by both a_t and a_s, c_1 the common concept carried by att_t and att_s, and c_2 the common concept carried by $x_t \in \mathcal{P}(att_t)$ and $x_s \in \nu(att_s)$. Moreover, let $\tau_{a,1}, \tau_{a,2}, \tau_{a,3}, \tau_{att,1}, \tau_{att,2}, \tau_{att,3}, \tau_{x,1}, \tau_{x,2}$, and $\tau_{x,3}$ be predefined thresholds. An ND-simulation relation of $\mathcal{B}_t$ by $\mathcal{S}$ is a relation $R \subseteq B_t \times S$ such that $\langle t, s \rangle \in R$ implies:

Extension of the largest ND-simulation relation in Sect. 5.2.4

1. if $t \in F_t$, then $s \in F$;
2. for all transitions $\langle t, a_t, t' \rangle \in \delta_t$:
   - there exists a transition $\langle s, a_s, k, s' \rangle \in \delta$;
   - for all transitions $\langle s, a_s, k, s' \rangle \in \delta$:
     - $sim(a_t, a_s) \geq \tau_{a,1}$ if reasoning is similarity,
     - $compat(a_t, a_s) \geq \tau_{a,2}$ if reasoning is compatibility,
     - $Sim(v_{s0}, v_{t0}, c_0) \geq \tau_{a,3}$ if reasoning is numerical;
   - for each attribute $att_t \in \alpha_t(a_t)$, there exists an $att_s \in \alpha_s(a_s)$:
     - $sim(att_t, att_s) \geq \tau_{att,1}$ if reasoning is similarity,
     - $compat(att_t, att_s) \geq \tau_{att,2}$ if reasoning is compatibility,
     - $Sim(v_{s1}, v_{t1}, c_1) \geq \tau_{att,3}$ if reasoning is numerical;
   - for all $att_t \in \alpha_t(a_t)$, $\langle t', s' \rangle \in R$ and:
     - $\nu(att_s) \in Fav$ if $\mathcal{P}(att_t) = \boldsymbol{Fav}$,
     - $\nu(att_s) \notin Dis$ if $\mathcal{P}(att_t) = \boldsymbol{Dis}$,
     - $\nu(att_s) \in Fav \cup Alt$ if $\mathcal{P}(att_t) = \boldsymbol{Fav/Alt}$,
     - $\nu(att_s) \notin Dis$ if $\mathcal{P}(att_t) = \boldsymbol{Fav/Dis}$,

Extension of the largest ND-simulation relation in Sect. 5.2.4 (continue)

- for each $x_t \in \mathcal{P}(att_t)$, there exists $x_s \in \nu(att_s)$ such that $sim(x_t, x_s) \geq \tau_{x,1}$ if $\mathcal{P}(att_t) = Similar(x_t)$,
- for each $x_t \in \mathcal{P}(att_t)$, there exists $x_s \in \nu(att_s)$ such that $compat(x_t, x_s) \geq \tau_{x,2}$ if $\mathcal{P}(att_t) = Compatible(x_t)$,
- for each numeric value $v_{t2} \in x_t$ and a common concept c_2, there exists $v_{s2} \in x_s$ such that $Sim(v_{s2}, v_{t2}, c_2) \geq \tau_{x,3}$ if $\mathcal{P}(att_t) = Numeric(v_{t2}, c_2)$.

Based on the revision in the largest ND-simulation relation, the controller generator should be extended to consider both resource reasoning and qualitative preferences conditions. The controller generator CG of $\mathcal{B}_t$ on $\mathcal{S}$ is $\langle \Sigma, A_t, A_s, I_n, \alpha_t, \xi, \omega \rangle$, where:

Extension of the controller generator defined in Sect. 5.2.4

1. $\Sigma = \{\langle t, s \rangle \in B_t \times S \mid t \preceq s\}$ is the set of CG states made by all pairs of $\mathcal{B}_t$ and $\mathcal{S}$ states that belong to the largest ND-simulation relation;
2. ξ is the transition relation, where $\sigma \xrightarrow{a_t, a_s, k} \sigma'$ in ξ, if and only if:
   - there is a transition $t \xrightarrow{a_t} t'$ in $\mathcal{B}_t$;
   - there is a transition $s \xrightarrow{a_s, k} s'$ in $\mathcal{S}$;
   - $sim(a_t, a_s) \geq \tau_{a,1}$ if reasoning is similarity;
   - $compat(a_t, a_s) \geq \tau_{a,2}$ if reasoning is compatibility;
   - $Sim(v_{s0}, v_{t0}, c_0) \geq \tau_{a,3}$ if reasoning is numerical;
   - for each attribute $att_t \in \alpha_t(a_t)$, there exists an $att_s \in \alpha_s(a_s)$:
     - $sim(att_t, att_s) \geq \tau_{att,1}$ if reasoning is similarity,
     - $compat(att_t, att_s) \geq \tau_{att,2}$ if reasoning is compatibility,
     - $Sim(v_{s1}, v_{t1}, c_1) \geq \tau_{att,3}$ if reasoning is numerical;

Extension of the controller generator defined in Sect. 5.2.4 (continue)

- for each attribute $att_t \in \alpha_t(a_t)$, there exists an $att_s \in \alpha_s(a_s)$:
  - $sim(att_t, att_s) \geq \tau_{att,1}$ if reasoning is similarity,
  - $compat(att_t, att_s) \geq \tau_{att,2}$ if reasoning is compatibility,
  - $Sim(v_{s1}, v_{t1}, c_1) \geq \tau_{att,3}$ if reasoning is numerical;
- for all $att_t \in \alpha_t(a_t)$:
  - $\nu(att_s) \in Fav$ if $\mathcal{P}(att_t) = \boldsymbol{Fav}$,
  - $\nu(att_s) \notin Dis$ if $\mathcal{P}(att_t) = \boldsymbol{Dis}$,
  - $\nu(att_s) \in Fav \cup Alt$ if $\mathcal{P}(att_t) = \boldsymbol{Fav/Alt}$,
  - $\nu(att_s) \notin Dis$ if $\mathcal{P}(att_t) = \boldsymbol{Fav/Dis}$,
  - for each $x_t \in \mathcal{P}(att_t)$, there exists $x_s \in \nu(att_s)$ such that $sim(x_t, x_s) \geq \tau_{x,1}$ if $\mathcal{P}(att_t) = Similar(x_t)$,
  - for each $x_t \in \mathcal{P}(att_t)$, there exists $x_s \in \nu(att_s)$ such that $compat(x_t, x_s) \geq \tau_{x,2}$ if $\mathcal{P}(att_t) = Compatible(x_t)$,
  - for each numeric value $v_{t2} \in x_t$ and a common concept c_2, there exists $v_{s2} \in x_s$ such that $Sim(v_{s2}, v_{t2}, c_2) \geq \tau_{x,3}$ if $\mathcal{P}(att_t) = Numeric(v_{t2}, c_2)$,
  - for all $\langle t'', s'' \rangle \in B_t \times S$, such that $s \xrightarrow{a_s, k} s''$ in $\mathcal{S}$ and $t \xrightarrow{a_t} t''$ in $\mathcal{B}_t$, it is the case that $\langle t'', s'' \rangle \in \Sigma$;

3. $\omega : \Sigma \times A_t \times A_s \to 2^{I_n}$ is the output function with $\omega(\sigma, a_t, a_s) = \{k \mid \exists \sigma' \in \Sigma$ such that $\langle \sigma, \langle a_t, a_s, k \rangle, \sigma' \rangle \in \xi\}$.

5.4 Contributions

In summary, this chapter makes the following contributions.

- It enhances the possibility of realizing a given target behavior due to an appropriate match between similar actions in the controller synthesis procedure. This

can be readily comprehended from the results of the experiment that evaluates the effect of similarity reasoning on the realized target behaviors.

– It extends the original behavior composition framework to support both qualitative and quantitative preferences. The conditions of the former discard some actions during controller synthesis. Given a set of candidate controllers, the latter can be considered for assigning a rank to each composition (see Chapter 6). To sum up, the benefit of such an extension lies in increasing the expressiveness of a target behavior with a better satisfaction level through control exercised on available behaviors.

– It introduces a hybrid framework for behavior composition to aggregate the benefits of both semantic-based and preference-based frameworks. Given such a framework, more matchable alternatives can be offered for a preference requested by a target behavior. Besides, with the aid of resource reasoning, a new quantitative preference model is defined over the attributes' values to choose the most similar one.

Chapter 6

Team Formation through Preference-Based Behavior Composition

The team formation problem in multiagent systems, where dynamic software components are agents, can be generalized to form a team of agents, which have explicit behaviors and whose tasks are equipped with multiple attributes. In this problem, the values of such attributes are compared with preferences attached to the desired tasks of a goal. A synthesized controller realizes the goal by invoking tasks of a subset of the available agents. Together they constitute a composition. Furthermore, utility values are assigned to the compositions and robustness is considered to be an important property of a team to prevent its deterioration when one or more of its agents fail. Finding a robust team that satisfies the goal's preferences with better utility values for compositions constitutes a difficult optimization problem. The proposed method to solve this problem consists in three phases summarized as follows.

- The controller synthesis with filtering on tasks with respect to qualitative preferences. The proposed solution revisits the largest ND-simulation relation as described in Sect. 5.2.4.
- The ranking of compositions based on their fitness with respect to the quantitative and qualitative preferences.
- The formulation of a new multiobjective optimization problem to take into account properties or utility values assigned to compositions and use of a mathematical programming technique to find solutions.

6.1 A Team Formation Problem

A typical team formation problem includes a set of agents $A = \{a_1, \ldots, a_n\}$; a set of tasks $T = \{t_1, \ldots, t_m\}$; a function $\tau : A \to 2^T$, which assigns to every agent a

subset of tasks that it can perform; and a goal $G \subseteq T$, which represents functional requirements [47]. In its simplest form, it corresponds to the set cover problem, which consists in identifying a subset C of A, with the smallest cardinality, such that C is an *effective team*, that is, $G \subseteq \cup_{a_i \in C} \tau(a_i)$ [24]. Such a team formation problem involves cooperation of agents according to the first condition stated in [27] (i.e., agents have a goal in common and their tasks tend to achieve that goal).

Various variants of this problem have been proposed in the past. For instance, the addition of a cost function $\kappa : A \to \mathbb{R}^+$, which assigns a cost to every agent, changes the objective to the minimization problem in order to determine an effective team with the minimal cost.[1] In an unpredictable environment or a hazardous system, a team must remain effective even if any k agents are removed from the original team. When a team satisfies this property, it is then said to be *k-robust* [47]. A bi-objective constraint-optimization problem then arises, because the cost must be minimized while k must be maximized. This variant adds an additional objective and intends to identify the Pareto-optimal front as illustrated in Example 6.1.1. More specifically, every team is such that there exists no other team with better cost and k value. More complex team formation problems can be defined when replacing the sets of tasks associated with agents by multisets of tasks [35] and considering more objectives. A common characteristic of all these variants is that most of them can be formulated as multiple-objective optimization problems [47] or linear integer programming problems [24]. Such a formulation is as follows.

$$\min[\sum_i \kappa(a_i) x_i, -K] \tag{1}$$

subject to

$$\sum_{i:t_j \in \tau(a_i)} x_i \geq K + 1; \tag{2}$$
$$x_i \in \{0, 1\}.$$

The two objectives of this problem are the total cost and degree of robustness

1. If $\kappa(a_i) = 1$ for all i, the optimization problem reduces to the set cover problem.

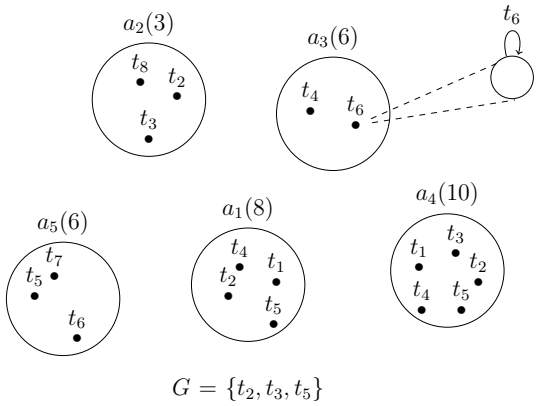


Figure 6.1: An instance of a team formation problem

(K). The Boolean variable x_i indicates the selection of agent a_i in the team and j ranges from 1 to $|G|$ (the number of tasks in the goal).

Other algorithm strategies have been used to conceive exact algorithms and heuristics for solving team formation problems: from greedy algorithms to evolutionary approximation algorithms, including planning, branch-and-bound, and dynamic programming algorithms [24]. All these problems find immediate application in the domain of cloud computing, where agents are virtual machines and tasks are microservices [28].

Example 6.1.1. In Fig. 6.1, $A = \{a_1, a_2, a_3, a_4, a_5\}$, $T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\}$, and $G = \{t_2, t_3, t_5\}$. The set of tasks of agent a_1 is $\tau(a_1) = \{t_1, t_2, t_4, t_5\}$ and its cost is $\kappa(a_1) = 8$. The following table gives some effective teams with their cost and degree of robustness.

Team	Cost	*k-robust*
$\{a_4\}$	10	0-robust
$\{a_1, a_2\}$	11	0-robust
$\{a_2, a_5\}$	9	0-robust
$\{a_1, a_2, a_4\}$	21	1-robust
$\{a_2, a_4, a_5\}$	19	1-robust

The Pareto-optimal front is $\{\{a_2, a_5\}, \{a_2, a_4, a_5\}\}$. Appendix F.1 provides the optimal solutions come from the linear programming package PuLP program. ■

6.2 Goal Realization via Preference-Based Behavior Composition

In an ordinary team formation problem, there are strong implicit assumptions about the tasks that an agent can perform:

– they can be invoked anytime and in any order;

– they are plain tasks, that is, without attributes or properties.

The combination of the behavior composition framework with a preference model, already presented in Sect. 5.2.4, can lead to an interesting variant of the team formation problem, which has, however, consequential impacts with regards to the solutions already proposed for forming a team, since mathematical regularity is lost. The benefit of such an approach lies in increasing the expressiveness of a goal with a better satisfaction level through control exercised on agents.

Formally, each agent (behavior) is represented by $\mathcal{B}_i = \langle B_i, T_i, \alpha_i, \delta_i, b_{i0}, F_i \rangle$, where T_i is a finite set of tasks and $\alpha_i : T_i \rightarrow 2^{Att}$ is a function assigning a subset of attributes of Att to every task. The rest of elements are the same as those defined in Sect. 5.2.4. The (enacted) system behavior is $\mathcal{S} = \langle S, T, I_n, \alpha, \delta, s_0, F \rangle$, where $T = \cup_i T_i$ is the set of tasks and $\alpha(t) \subseteq \alpha_i(t)$ for $t \in T_i$ (only common attributes of a task may be considered). For the other elements of enacted system behavior see Sect. 5.2.4. Furthermore, the goal (target behavior) is $\mathcal{B}_t = \langle B_t, T_t, \alpha_t, \delta_t, b_{t0}, F_t \rangle$ with $T_t \subseteq T$ and $\alpha_t(t) \subseteq \alpha(t)$ for $t \in T_t$. The attributed multiagent system is $\langle a_1, \ldots, a_n \rangle$, where a_i has behavior $\mathcal{B}_i$ $(1 \leq i \leq n)$.

The revisited largest ND-simulation relation proposed in Sect. 5.2.4 with filtering based on the qualitative preferences is applied for the controller generator synthesis. From the controller generator, the set of controllers can be extracted. More precisely, a controller P is a subtransition system of the controller generator with the following restrictions. There is at most one transition from each state on a given task (while preserving nondeterminism) and this task is delegated to only one agent. This corresponds to Def. 2.2.4.

Given a controller, a composition can be defined as follows.

Definition 6.2.1. A composition, denoted $C = \langle P, \{a_{i_1}, \ldots, a_{i_l}\}\rangle$, is the realization of the goal by a controller P together with the subset of agents $\{a_{i_1}, \ldots, a_{i_l}\} \subseteq A$ ($l \leq n$), where at least one task is delegated to a_{i_j} for every $1 \leq j \leq l$.

This definition is different from the one introduced in the original behavior composition framework as explained in Sect. 2.2.1.

6.3 Formulation a New Team Formation Problem

The team formation problem through preference-based behavior composition is formulated as follows:

> Given an attributed multiagent system $\mathcal{S} = \langle a_1, \ldots, a_n\rangle$ and a deterministic goal G with preferences, find a set of compositions $\{C_1, \ldots, C_m\}$, such that each composition $C_i = \langle P_i, \{a_{i_{j_1}}, \ldots, a_{i_{j_l}}\}\rangle$ realizes (through P_i) the goal G and presents the best match for the preferences, and agents involved in these compositions form the more robust team at less cost.

The original behavior composition problem is then extended to the one of finding the more robust team at less cost (i.e., the utility factors are cost and degree of robustness). It comprises distinct parts. Each of them may be declined in numerous variants. One variant is further formalized with the following multiobjective optimization problem, in which the variable x_j, y_i, and x_{ij} are Boolean variables and K is an integer variable:

$$\min[\sum_i \kappa_1(C_i)y_i + \sum_j \kappa_2(a_j)x_j + \sum_{i,j} \kappa_3(a_j, C_i)x_{ij}, -K] \tag{1}$$

subject to

$$\sum_i x_{ij} \geq x_j; \tag{2}$$

$$\sum_{a_j \in C_i} x_{ij} = |C_i| y_i; \qquad \sum_{a_j \notin C_i} x_{ij} = 0; \tag{3}$$

$$\sum_{j:t_k \in \hat{\tau}(a_j)} x_j \geq K + 1; \tag{4}$$

$$y_i, x_j, x_{ij} \in \{0, 1\};$$

$$C_i \text{ realizes } G \text{ at best.} \tag{5}$$

The two objectives of this problem concern costs (κ_1, κ_2, κ_3) and degree of robustness (K), respectively. The function κ_1 gives the global cost of a composition. The functions κ_2 and κ_3 represent the access cost to an agent and connection cost of an agent to a composition, respectively. The variables i, j, and k range from 1 to m (the number of candidate compositions), from 1 to n (the number of agents), and from 1 to $|T_t|$ (the number of tasks in the goal), respectively. There are three blocks of constraints. Constraints (2) relate the individual agents to the corresponding agents in the compositions. If agent a_j is selected (i.e., $x_j = 1$ in the solution), then a_j belongs to at least one composition, say C_i (i.e., $x_{ij} = 1$ for at least one i in the solution). Constraints (3) ensure that each composition includes only their own agents. The term $|C_i|$ denotes the number of agents in composition C_i. If composition C_i is selected (i.e., $y_i = 1$ in the solution), then its agents are necessary attached to it (i.e., for all j such that $a_j \in C_i$, $x_{ij} = 1$ in the solution). These two subblocks of constraints could also be written as follows: $\sum_j In(a_j, C_i) x_{ij} = |C_i| y_i$, where the predicate In holds whether $a_j \in C_i$. Constraints (4) correlate the degree of robustness and the number of agents that can perform the same task. The function $\hat{\tau}$ is a restriction of τ. It gives only the set tasks delegated to an agent, not all the tasks it can perform. For a given task t_k of the goal, if it is also a task of agent a_j and it is delegated to a_j then it counts for one whether a_j is selected (i.e., $x_j = 1$ in the solution). Without loss of generality, it is assumed that $\hat{\tau}(a_i) \cap \hat{\tau}(a_j) = \emptyset$, for all distinct pairs of agent a_i, a_j that belong to the same composition (care must be taken when considering a task delegated to more than one agent in the same composition). Due to the last

constraint (5), the problem rests on another multiobjective optimization problem, which determines a subset of the set of candidate compositions that meet the preferences of the goal to the greatest extent possible. Its solution cannot be computed by exploiting usual mathematical programming techniques, since the maximization is over an unknown set of compositions, which must be generated while taking into account some qualitative preferences as described in Sect. 5.2.4. Several heuristics and an efficient nondominated sort algorithm [25] are used instead for approximate ranking of compositions.

6.3.1 Nondominated Sorting

In the case that there are multiple objectives in a problem, instead of a single optimal solution, a set of optimal ones, known as the Pareto-optimal solutions, can be obtained. Given such a problem, a fast nondominated sorting algorithm, presented in Algorithm 3, has been proposed to discover the Pareto-optimal solutions [25].

In this algorithm, for determining the first nondominated front in a set of solutions, each solution is compared with every other solution in the set to explore whether it is dominated. The procedure is repeated to discover all candidates of the first nondominated front in the set, namely the solutions having the rank 1 (lines 2–16). Then, for finding the next nondominated front, the solutions of the first front are temporarily ignored and the procedure is repeated again (lines 20–28). For each solution $x \in X$, the number of solutions dominating x is denoted by n_x in the algorithm. The set S_x includes the solutions that x dominates. Besides, the set F_i contains the solutions having the rank i, and the set Y stores the members of the next front.

Algorithm 3 Fast nondominated sort (X)

```
1: Let X be the set of all solutions
2: for each x ∈ X do
3:     S_x ← ∅
4:     n_x ← 0
5:     for each y ∈ X do
6:         if (x ≺ y) then
7:             S_x ← S_x ∪ {y}
8:         else if (y ≺ x) then
9:             n_x ← n_x + 1
10:        end if
11:    end for
12:    if (n_x = 0) then
13:        x_rank ← 1
14:        F_1 ← F_1 ∪ {x}
15:    end if
16: end for
17: i ← 1
18: while F_i ≠ ∅ do
19:    Y ← ∅
20:    for each x ∈ F_i do
21:        for each y ∈ S_x do
22:            n_y ← n_y − 1
23:            if (n_y = 0) then
24:                y_rank ← i + 1
25:                Y ← Y ∪ {y}
26:            end if
27:        end for
28:    end for
29:    i ← i + 1
30:    F_i ← Y
31: end while
```

The computational complexity of the algorithm is $O(m \times n^2)$, where m is the number of objectives and n is the number of all solutions.

6.3.2 Composition Ranking

Given a set of compositions such that the associated controllers are synthesized based only on the qualitative preferences as presented in Sect. 5.2.4, it is impossible to distinguish between the compositions satisfying favorite preferences and those

Table 6.1: Rating of preferences

Composition	Airplane type	Airplane price	Hotel type	Hotel price	Meals type	Meals price	Bus type	Bus price
C_1	0	1	1	0	1	1	1	1
C_2	0	1	0	1	1	1	1	1
C_3	0	0	0	1	1	1	1	1
C_4	0	0	1	0	1	1	1	1
C_5	0	0	0	0	1	1	1	1
C_6	0	1	0	0	1	1	1	1

offering just alternative or non-dislike preferences due to their semantics and implementation details. A similar remark applies for the quantitative preferences. The compositions with the most desirable values cannot be identified during the generation of compositions. An additional step is required to classify compositions in a number of ranks when comparing all attribute values of their tasks with respect to the corresponding preferences of the goal. The goal here is not only to solve the optimization problem (5), but to collect and rank a large number of compositions. First, a composite preference on attributes of every task, namely Pareto (balanced), prioritized, and weighted (numerical), is applied to reflect their relative importance. Second, rating values are associated with the relevant attributes based on the goal's preferences. Finally, the rating values are aggregated into one (total value) or more (Pareto) values to determine the rank of each composition.

Example 6.3.1. The six compositions identified in Example 5.2.2 have been ranked with respect to the Pareto composite preference for attributes of each task, Boolean values (1 for the best, 0 otherwise) as rating values, and summation over them. The final rating values of C_1 to C_6 are 6, 6, 5, 5, 4, and 5, respectively. Table 6.1 provides the details about the rating of preferences. For instance, in C_1, the rating value for hotel type is 1, as it satisfies the favorite preference of the goal (`4-star` hotel). However, this value is 0 for C_2, since the `3-star` hotel is an alternative preference. The rating value is also 0 for seat class of the airplane flight (airplane type), since it is a dislike preference that has been filtered during the generation of compositions considered in constraints (5). The rating values for the rest of preferences in the

Table 6.2: The number of compositions realizing goals

Number of tasks	Sound compositions	Selected compositions	Potential compositions
$p = 2$	2	5	7
$p = 3$	5	109	127
$p = 4$	15	32297	32767
$p = 5$	52	$\approx 2.1 \times 10^9$	$2^{31} - 1$

composition are 1, because all the favorite preferences are satisfied.

Given the final rating values and Algorithm 3, the compositions C_1 and C_2 have rank 1, C_3, C_4 and C_6 have rank 2, and C_5 has rank 3. ■

Example 6.3.2. Let $\kappa_1(C_i) = 1$ (for all i), $\kappa_2(a_j) = 1$ (for all j), and $\kappa_3(a_j, C_i) = 1$ (for all i, j) in the optimization problem defined by (1) to (5). Then, the total cost of compositions C_1, C_2, C_3 and C_4 is 5, and is 7 for C_5, and C_6. For instance, because there exists two agents (a_2, a_3) in the composition C_1, the total cost is calculated: $\kappa_1(C_1) + \kappa_2(a_2) + \kappa_2(a_3) + \kappa_3(a_2, C_1) + \kappa_3(a_3, C_1) = 5$. If the compositions with rank 1 or 2, obtained by the nondominated sort, are those considered in (5), then the Pareto front (best solutions) are C_1, C_2, C_3, and C_4. The agent team is $\{a_2, a_3\}$ and it is 0-robust. Appendix F.2 provides the PuLP program of the example. ■

6.4 Experiments with a Synthetic Problem

The synthetic problem consists in n available agents that carry out tasks, ranging from groups of agents having one task to those having $p-1$ different tasks, including the group of a single agent having p different tasks, where p is the total number of tasks in the goal. Each agent executes sequentially all its tasks in the same order as those in the goal. The number of agents is $n = \Sigma_{k=1}^{p} \binom{p}{k}$. The goal includes the p tasks requested sequentially.

Table 6.2 provides the details about the number of compositions computed from three strategies for the goals, whose number of tasks vary from 2 to 5. In the first strategy (sound compositions), if a composition has more than one agent, the intersection of the tasks of its agents should be empty. The second strategy (selected compositions) is, however, less strict, since each composition can involve some agents

such that the intersection of their tasks is not empty. Finally, the last strategy considers all combinations of agents. So, the total number of potential compositions is $2^{2^p-1} - 1$. Some of them do not, however, realize the goal.

Appendix F.3 provides the detail about a Ruby program, which computes the number of compositions according to these strategies.

To simplify the experiments, a sample of 100 sound compositions was considered among those possible for two different number of tasks, namely $p = 7$ and $p = 8$. Each task has three attributes. One has a numerical value, ranging from 1 to 10 and chosen randomly. It corresponds to any sort of quantitative preference. The others have lexical terms as domains and correspond to favorite/alternative or favorite/dislike preferences. A favorite term is represented by 1 and a non-favorite or alternative term by 0. The 0 and 1 are generated randomly with the constraint that 40 % of them be 1 (favorite). The Pareto heuristic was implemented by using a nondominated sort. Figures 6.2 and 6.3 provide the results computed by a Ruby program for $p = 7$ and $p = 8$, respectively. Results are averages calculated after ten iterations with different random data.

As depicted in the left graphs of the both figures, the horizontal axes indicate the ranks, ranging from 1 to 30 and from 1 to 31 in Fig. 6.2 and Fig. 6.3, respectively. The vertical axes show the number of compositions. Bell-shaped curves arise for the three composite preferences (Pareto, prioritized, and weighted). The number of best and worst compositions (i.e., the most and least matchable with goal preferences, respectively) are at the extremities of the curves. They look like a normal distribution with heavier tails in the case of Pareto and prioritized preferences (e.g., like a logistic distribution), since attributes values have been generated randomly. The compositions are distributed in nine and ten groups with the Pareto preference for $p = 7$ and $p = 8$, respectively. In both figures, the one of rank 1 contains in average more compositions in comparison with the other groups of rank 1 for the prioritized and weighted preferences. The fluctuation among groups is less variable for the weighted preference, which tends to discriminate compositions between a larger range of ranks. The reason for that is that the prioritized preference gives importance to one attribute per task, the Pareto preference gives the same priority to all the attributes, and the weighted preference takes also into account all the attributes, but assigns different

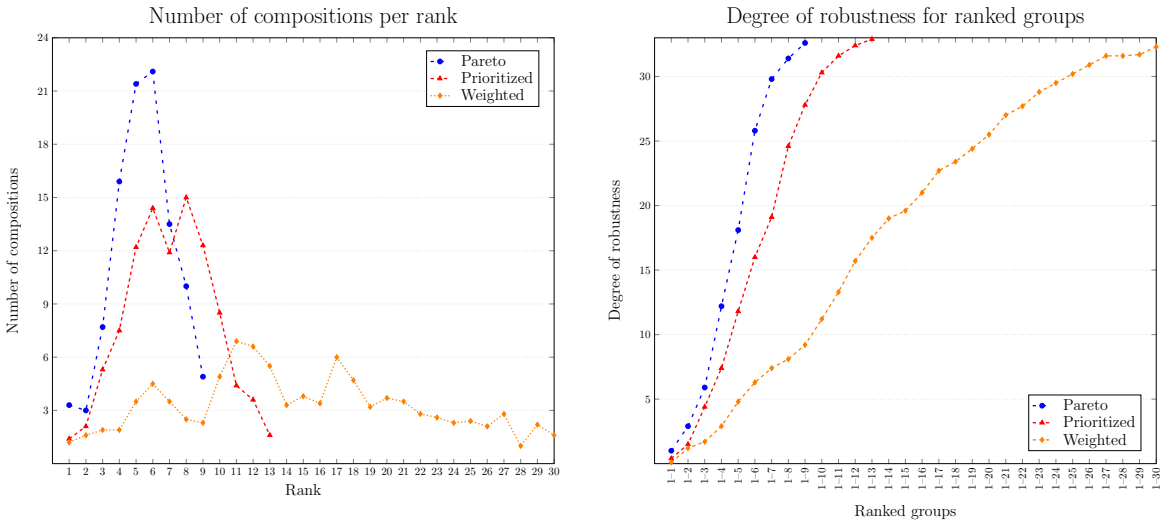


Figure 6.2: Experimental results with $p = 7$

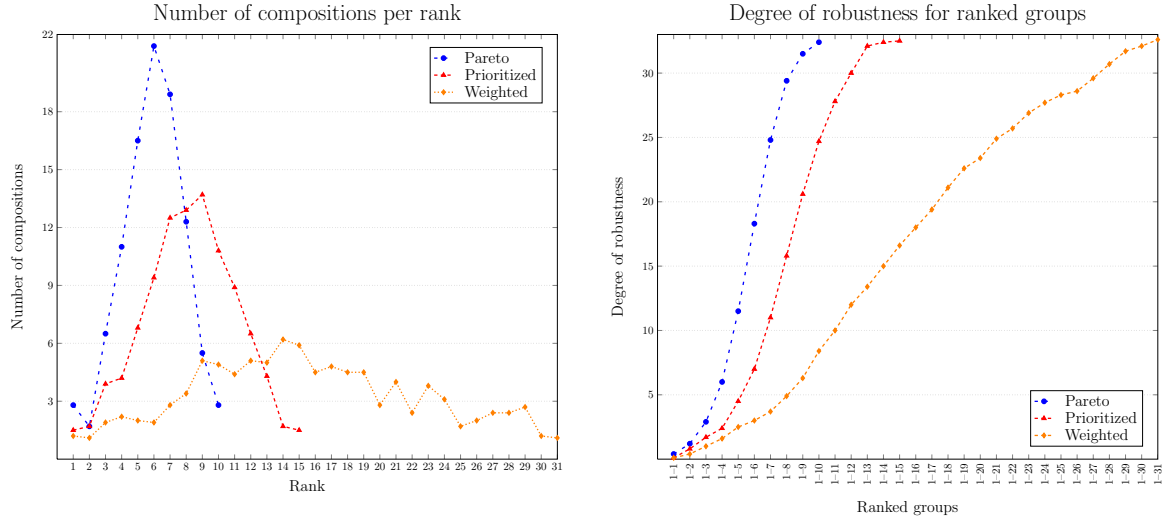


Figure 6.3: Experimental results with $p = 8$

priorities (or weights) to them. Furthermore, a comparison between Fig. 6.2 and Fig. 6.3 shows when the number of tasks increases, the number of ranks increases and the number of compositions per each rank decreases. In fact, an increase in the number of tasks leads to the distribution of compositions in a larger range of ranks. More experiments are, however, required to confirm these trends.

The right graphs of the both figures show the degree of robustness for groups of compositions. With respect to the horizontal axis, each label of the scale has the format "1–i" and refers to all compositions with a rank between 1 and i. It can be observed from the figures that the slopes of the curves are high, except at the beginning and end. The slope is higher for the Pareto heuristic and lower for the weighted heuristic. This is consistent with the bell-shaped curves in the left graphs, in particular the shape of the curves for the Pareto and prioritized heuristics is quite similar. Compared with the Pareto heuristic, more ranks and thus more groups of compositions, with a larger gap between the first and last, must be added to the team to reach the same degree of robustness. Finally, it is impossible to achieve a reasonable degree of robustness when considering only compositions of rank one.

6.5 Contributions

In summary, this chapter makes the following contributions.

- It extends the basic robust team formation problem to include agents with behaviors, tasks with attributes, and scalar-valued functions over compositions.
- It uses the preference-based framework of behavior composition for synthesizing the compositions.
- It considers a large number of compositions (not only one plan) without which it is impossible to cope with the Pareto preference or produce robust teams.
- It takes into account more global properties on compositions for discrimination at a higher level in order to formulate multiobjective optimization problems. This solution improves the one proposed in [10].
- It provides experiments with a synthetic multiagent system to show the impacts of composite preferences (or heuristics) on the distribution of compositions with respect to their rank and on the degree of robustness of a team formed from agents.

Conclusion

This conclusion provides a summary of the contributions of this thesis. Nevertheless, some of such contributions are not a panacea; further work should be done to fully achieve all the objectives aforementioned in the thesis introduction.

A Review of Contributions

In the case of dynamic software components as online services, new abstract models with explicit control were introduced for IoT and haptic devices, which are, respectively, classified into the second and third generations of service representations. The behavior of such components were described with a usage protocol to provide a foundation for synthesis and verification in the service-oriented IoT middleware. Then, an architecture was designed to construct composite services in such middleware through which not only the abstract models were adopted, but the explicit controllers inside of composite services were automatically synthesized via the behavior composition framework. Two possible scenarios were introduced for constructing composite services in middleware. In one scenario, the necessary condition for constructing a composite service was the satisfaction of real-time constraints imposed by the target service. Hence, some verifications were performed on the interactions in a composite service through Uppaal to check possible time-actionlocks before the delivery of the composite service to the end users. In the other scenario, composite services were constructed without considering possible deadlines for the target services. This scenario allowed end users to verify composite services with respect to several TCTL formulas in Uppaal. Such verifications enabled the users to choose the composite services that satisfy their feasible real-time requirements.

Three scenarios were introduced for the case of a huge number of composite services executed simultaneously. The first was the creation of individual threads of available atomic services used by each composite service. The second permitted the composite services to have a synchronization on a unique available atomic service with respect to a common set of operations. In the third scenario, the composite services were synchronized on distinct available atomic services. Contrary to the first

scenario, which is more appropriate for the cases in which the operations of atomic services are virtual, the others can be used when the atomic services are physical or the access to their operations is exclusive, analogous to haptic devices. These scenarios were formalized according to a process calculus, which reformulates the elements of the behavior composition framework as process terms. Furthermore, some actions with new operational semantic rules were introduced for possible deadlocks in the last two scenarios.

Advances in service-oriented computing systems and IoT middleware have led to the consideration of more characteristics for the operations of atomic and composite services. One characteristic is the calculation of the degree of match between service operations with regards to the resource reasoning functions. To this end, the algorithms for synthesizing the controllers of composite services were revisited to improve the possibility of realization of target services through a more flexible match between operations. Another characteristic is the attachment of multiple attributes to the atomic service operations and the enrichment of the implementation of a composite service to expose requirement preferences. By doing so, both the notions of largest ND-simulation relation and the controller generator in the original behavior composition framework were revisited to filter the atomic service operations with respect to the qualitative atomic preferences. Furthermore, when the attribute values and preferences are numeric values, such as cost, the composite services are ranked according to Pareto, prioritized, and weighted preferences. Such ranks enable the end users to readily select their desirable composite services.

In the case of dynamic software components as agents, a team formation problem was introduced through the integration of a preference-based behavior composition framework in which a composition involving an effective team of agents was constructed to realize a common goal whose tasks expose some atomic preferences. For the discrimination of compositions at a higher level, some global parameters were taken into account with the compositions to formulate an optimization problem. Then, this problem was solved with a mathematical programming technique and implemented by the package PuLP.

Criticism of the Obtained Results

In the real-time formulation of the behavior composition framework, the time constraints were ignored during the step of controller synthesis in SMV/TLV. Then, the simulation and verification of each synthesized controller were achieved through another model-checking tool, namely UPPAAL. However, a revision in the controller synthesis algorithm of the original framework should lead to the elimination of the verification step via UPPAAL, since such a verification and simulation is time consuming, especially when the controllers of the synthesis step cannot realize the real-time constraints imposed by the target service and available atomic services.

In the semantic-based behavior composition framework, the values, which are returned from the resource reasoning functions and show the degree of match between service operations, cannot be computed by SMV/TLV. These values were manually computed and inserted as integer numbers in the SMV module skeletons extended for the implementation of the semantic-based framework. Hence, this problem may raise a question about how to establish a link between SMV/TLV and the available ontological engineering tools for building ontology graphs and knowledge-based solutions.

When integrating the behavior composition framework into the robust team formation problem, recovery procedures must be defined to manage failures in a k-robust team. Some indications are given in [32], but they do not consider, for instance, the reversibility of tasks or any other assumptions on agents. Furthermore, contrary to flat agents and tasks, compositions reveal properties at three levels: properties associated with compositions, agents, and tasks. Although a solution to this problem was suggested (with no other utility factors than cost and robustness), a uniform solution must be developed for both the preference model and synthesis procedures. Moreover, centralized control is generally too restrictive with respect to the multiagent paradigm. Theories should be developed with synthesis algorithms to generate control policies as well as negotiation policies, which could be distributed into agents. The former is more appropriate for orchestration and the latter for choreography. Finally, when each of a huge number of compositions involves a team of agents, finding a more robust composition with less cost is an NP-hard problem.

Future Work

With the growth in the number of services supplied by many service providers of service-oriented computing systems located in different geographically areas, the significance of reaching a decentralized control mechanism as a solution for the service composition problem is totally obvious. To this end, the centralized approach introduced in the behavior composition framework must be replaced to include several decentralized controllers. Although there exists a contribution that investigated the problem of a decentralized controller with respect to the behavior composition framework [55], communication among controllers relies on message exchanges via a common channel, and the reliability of the communication is not a concern. Moreover, in this solution, there is no strategy in which the available atomic services provide operations with very different response times. Regarding the relationships between the supervisory control theory and behavior composition framework [9], this question suggests that other relationships between these two frameworks can be further explored, particularly in the area of decentralized control.

A hierarchical control was not taken into consideration in the original behavior composition framework. However, when the behaviors were considered as part of dynamic software components, they were used both as interfaces and implementations. The interface is an abstraction of the implementation, which hides implementation operations. Hence, two types of operation, namely, internal and external operations, must be taken into consideration. Both belong to the implementation, but the latter also appears in the interface. This view of components can lead to hierarchical control through which a component is able to hierarchically interact with other components to realize a desired specification. So, the extension of the original framework to support such kinds of control is another potential research avenue for future consideration.

Last but not least, the investigation of the criticisms and problems mentioned in the preceding section and the provision of reasonable solutions for them constitute the rest of future work from this thesis.

Appendix A

A.1 Terminologies of Elements in Different Chapters

Table A.1: The elements of behavior composition framework in Chapters 1 to 3

Original framework	Chapter 1	Chapter 2	Chapter 3
Action	Operation	Action	Operation
Available behavior	Available atomic service	Available behavior	Available atomic service
Target behavior	Target service	Target behavior	Target service
Controller generator		Controller generator	Orchestrator generator
Generated controller Composition	Orchestrator	Generated controller Composition	Orchestrator

Table A.2: The elements of behavior composition framework in Chapters 4 to 6

Original framework	Chapter 4	Chapter 5	Chapter 6
Action	Operation	Action Resource	Task
Available behavior	Available component Interface	Available behavior	Available agent
Target behavior	Implementation	Target Behavior	Goal
Controller generator		Controller generator	Controller generator
Generated controller Composition	Orchestrator	Generated controller	Controller
	Composition		Composition

Appendix B

B.1 SMV Code of Example 2.1.1 with Target $\mathcal{B}_{t1}$

```
MODULE main
 VAR
   sys: system Sys(ctr.index);
   ctr: system Ctr;
 DEFINE
   good := (ctr.initial & sys.initial) | !(sys.failure);

MODULE Ctr
 VAR
   index : 0..4;
 INIT
   index = 0
 TRANS
   case
    index=0 : next(index)!=0;
    index!=0 : next(index)!=0;
   esac
 DEFINE
   initial := (index=0);

MODULE Sys(index)
 VAR
  operation : {start_op,archive,publish,put-caption,translate,upload-photo,upload-video,write-story};
  target : Target(operation);
  B1 : Behavior1(index,operation);
  B2 : Behavior2(index,operation);
  B3 : Behavior3(index,operation);
  B4 : Behavior4(index,operation);
 DEFINE
  initial := (B1.initial & B2.initial & B3.initial & B4.initial & target.initial & operation=start_op);
  failure := (B1.failure |B2.failure |B3.failure |B4.failure) |
             (target.final & !(B1.final & B2.final & B3.final & B4.final));

----------Target behavior #1----------

MODULE Target(op)
 VAR
   state : {start_st,t1,t2,t3,t4};
 INIT
   state = start_st & op = start_op
 TRANS
  case
    state = start_st & op = start_op : next(state) = t1 & next(op) = write-story;
    state = t1  & op = write-story  : next(state) = t2 & next(op) in {translate} ;
    state = t2  & op = translate  : next(state) = t3 & next(op) = archive ;
    state = t3  & op = archive : next(state) = t4 & next(op) = publish ;
    state = t4  & op = publish : next(state) = t1 & next(op) = write-story ;
  esac
 DEFINE
   initial := state=start_st & op=start_op;
   final := state in {t1};

----------Available behavior #1----------

MODULE Behavior1(index,operation)
 VAR
   state : {start_st,a1,a2,a3};
 INIT
   state=start_st
 TRANS
   case
    state=start_st & operation=start_op & index=0: next(state)=a1;
    (index != 1) : next(state) = state;
    (state=a1 & operation = upload-video) : next(state) in {a2};
    (state=a2 & operation = archive) : next(state) in {a3};
    (state=a2 & operation = put-caption) : next(state) in {a3};
    (state=a3 & operation = archive) : next(state) in {a1};
   esac
 DEFINE
   initial := state=start_st & operation=start_op  & index = 0 ;
   failure := index = 1 & !( (state = a1 & operation in {upload-video}) |
              (state = a2 & operation in {archive,put-caption}) | (state = a3 & operation in {archive}));
   final := state in {a1,a2,a3};

----------End of available behavior #1----------
```

```
----------Available behavior #2----------

MODULE Behavior2(index,operation)
 VAR
   state : {start_st,b1,b2,b3};
 INIT
   state=start_st
 TRANS
   case
    state=start_st & operation=start_op & index=0 : next(state)=b1;
    (index != 2) : next(state) = state;
    (state=b1 & operation = write-story) : next(state) in {b2};
    (state=b2 & operation = translate) : (next(state) in {b1} | next(state) in {b3});
    (state=b3 & operation = archive) : next(state) in {b1};
   esac
 DEFINE
   initial := state=start_st & operation=start_op & index = 0 ;
   failure := index = 2 & !( (state = b1 & operation in {write-story}) | (state = b3 & operation in {archive}) |
              (state = b2 & operation in {translate}));
   final := state in {b1,b3};

----------End of available behavior #2----------

----------Available behavior #3----------

MODULE Behavior3(index,operation)
 VAR
  state : {start_st,c1,c2};
 INIT
  state=start_st
 TRANS
   case
    state=start_st & operation=start_op & index=0 : next(state)=c1;
    (index != 3) : next(state) = state;
    (state=c1 & operation = archive) : next(state) in {c1};
    (state=c1 & operation = upload-video) : next(state) in {c2};
    (state=c2 & operation = archive) : next(state) in {c1};
    (state=c1 & operation = publish) : next(state) in {c1};
   esac
 DEFINE
   initial := state=start_st & operation=start_op  & index = 0;
   failure := index = 3 & !( (state = c1 & operation in {archive, publish, upload-video}) |
              (state = c2 & operation in {archive}));
   final := state in {c1,c2};

----------End of available behavior #3----------

----------Available behavior #4----------

MODULE Behavior4(index,operation)
 VAR
  state : {start_st,d1,d2,d3};
 INIT
  state=start_st
 TRANS
   case
    state=start_st & operation=start_op & index=0 : next(state)=d1;
    (index != 4) : next(state) = state;
    (state=d1 & operation = upload-photo) : next(state) in {d2};
    (state=d2 & operation = put-caption) : next(state) in {d3};
    (state=d3 & operation = translate) : next(state) in {d1};
   esac
 DEFINE
   initial := state=start_st & operation=start_op & index = 0;
   failure := index = 4 & !( (state = d1 & operation in {upload-photo}) |
              (state = d2 & operation in {put-caption}) | (state = d3 & operation in {translate}));
   final := state in {d1,d3};

----------End of available behavior #4----------

---------------TLV output (Controller generator #1)---------------

Check Realizability
Specification is realizable
Check that a symbolic strategy is correct
Transition relation is complete
All winning states satisfy invariant
Automaton States

State 1
 sys.operation = start_op      sys.target.state = start_st       sys.B1.state = start_st
 sys.B2.state = start_st       sys.B3.state = start_st           sys.B4.state = start_st
 ctr.index = 0

State 2
 sys.operation = write-story      sys.target.state = t1          sys.B1.state = a1
 sys.B2.state = b1                sys.B3.state = c1              sys.B4.state = d1
```

```
 ctr.index = 2


State 3
 sys.operation = translate     sys.target.state = t2            sys.B1.state = a1
 sys.B2.state = b2             sys.B3.state = c1                sys.B4.state = d1
 ctr.index = 2

State 4
 sys.operation = archive       sys.target.state = t3            sys.s1.state = a1
 sys.s2.state = b3             sys.s3.state = c1                sys.s4.state = d1
 ctr.index = 2

State 5
 sys.operation = archive       sys.target.state = t3            sys.s1.state = a1
 sys.s2.state = b1             sys.s3.state = c1                sys.s4.state = d1
 ctr.index = 3

State 6
 sys.operation = publish       sys.target.state = t4            sys.s1.state = a1
 sys.s2.state = b1             sys.s3.state = c1                sys.s4.state = d1
 ctr.index = 3

Automaton Transitions
 From 1 to  2
 From 2 to  3
 From 3 to  4 5
 From 4 to  6
 From 5 to  6
 From 6 to  2

Automaton has 6 states, and 7 transitions
user time: 0.016 s
BDD nodes allocated: 10037
max amount of BDD nodes allocated: 10037
Bytes allocated: 655424
```

B.2 SMV Code of Example 2.1.1 with Target $\mathcal{B}_{t2}$

```
MODULE main
 VAR
   sys: system Sys(ctr.index);
   ctr: system Ctr;
 DEFINE
   good := (ctr.initial & sys.initial) | !(sys.failure);

MODULE Ctr
 VAR
   index : 0..4;
 INIT
   index = 0
 TRANS
   case
    index=0 : next(index)!=0;
    index!=0 : next(index)!=0;
   esac
 DEFINE
   initial := (index=0);

MODULE Sys(index)
 VAR
  operation : {start_op,archive,publish,put-caption,translate,upload-photo,upload-video,write-story};
  target : Target(operation);
  B1 : Behavior1(index,operation);
  B2 : Behavior2(index,operation);
  B3 : Behavior3(index,operation);
  B4 : Behavior4(index,operation);
 DEFINE
  initial := (B1.initial & B2.initial & B3.initial & B4.initial & target.initial & operation=start_op);
  failure := (B1.failure |B2.failure |B3.failure |B4.failure) |
             (target.final & !(B1.final & B2.final & B3.final & B4.final));

----------Target behavior #2----------
MODULE Target(op)
 VAR
  state : {start_st,t1,t2,t3,t4};
 INIT
  state = start_st & op = start_op
 TRANS
   case
     state = start_st & op = start_op : next(state) = t1 & (next(op) = (upload-photo) | next(op) = (upload-video));
     state = t1  & op = upload-video  : next(state) = t4 & next(op) in {archive};
     state = t1  & op = upload-photo  : next(state) = t2 & next(op) in {put-caption};
     state = t2  & op = put-caption   : next(state) = t3 & next(op) in {translate};
     state = t3  & op = translate : next(state) = t4 & next(op) in {archive} ;
```

```
     state = t4  & op = archive : next(state) = t1 & (next(op) = (upload-photo) | next(op) = (upload-video));
   esac

 DEFINE
   initial := state=start_st & op=start_op;
   final := state in {t1};

----------Available behavior #1-----------

MODULE Behavior1(index,operation)
 VAR
   state : {start_st,a1,a2,a3};
 INIT
   state=start_st
 TRANS
   case
    state=start_st & operation=start_op & index=0: next(state)=a1;
    (index != 1) : next(state) = state;
    (state=a1 & operation = upload-video) : next(state) in {a2};
    (state=a2 & operation = archive) : next(state) in {a3};
    (state=a2 & operation = put-caption) : next(state) in {a3};
    (state=a3 & operation = archive) : next(state) in {a1};
   esac
 DEFINE
   initial := state=start_st & operation=start_op  & index = 0 ;
   failure := index = 1 & !( (state = a1 & operation in {upload-video}) |
              (state = a2 & operation in {archive,put-caption}) | (state = a3 & operation in {archive}));
   final := state in {a1,a2,a3};

----------End of available behavior #1----------




----------Available behavior #2----------

MODULE Behavior2(index,operation)
 VAR
   state : {start_st,b1,b2,b3};
 INIT
   state=start_st
 TRANS
   case
    state=start_st & operation=start_op & index=0 : next(state)=b1;
    (index != 2) : next(state) = state;
    (state=b1 & operation = write-story) : next(state) in {b2};
    (state=b2 & operation = translate) : (next(state) in {b1} | next(state) in {b3});
    (state=b3 & operation = archive) : next(state) in {b1};
   esac
 DEFINE
   initial := state=start_st & operation=start_op & index = 0 ;
   failure := index = 2 & !( (state = b1 & operation in {write-story}) | (state = b3 & operation in {archive}) |
              (state = b2 & operation in {translate}));
   final := state in {b1,b3};

----------End of available behavior #2----------

----------Available behavior #3----------

MODULE Behavior3(index,operation)
 VAR
  state : {start_st,c1,c2};
 INIT
  state=start_st
 TRANS
   case
    state=start_st & operation=start_op & index=0 : next(state)=c1;
    (index != 3) : next(state) = state;
    (state=c1 & operation = archive) : next(state) in {c1};
    (state=c1 & operation = upload-video) : next(state) in {c2};
    (state=c2 & operation = archive) : next(state) in {c1};
    (state=c1 & operation = publish) : next(state) in {c1};
   esac
 DEFINE
   initial := state=start_st & operation=start_op  & index = 0 ;
   failure := index = 3 & !( (state = c1 & operation in {archive, publish, upload-video}) |
              (state = c2 & operation in {archive}));
   final := state in {c1,c2};

----------End of available behavior #3----------

----------Available behavior #4----------

MODULE Behavior4(index,operation)
 VAR
  state : {start_st,d1,d2,d3};
 INIT
  state=start_st
 TRANS
```

```
   case
    state=start_st & operation=start_op & index=0 : next(state)=d1;
    (index != 4) : next(state) = state;
    (state=d1 & operation = upload-photo) : next(state) in {d2};
    (state=d2 & operation = put-caption) : next(state) in {d3};
    (state=d3 & operation = translate) : next(state) in {d1};
   esac
 DEFINE
   initial := state=start_st & operation=start_op & index = 0 ;
   failure := index = 4 & !( (state = d1 & operation in {upload-photo}) |
              (state = d2 & operation in {put-caption}) | (state = d3 & operation in {translate}));
   final := state in {d1,d3};

----------End of available behavior #4----------

---------------TLV output (Controller generator #2)---------------

Check Realizability
Specification is realizable
Check that a symbolic strategy is correct
Transition relation is complete
All winning states satisfy invariant
Automaton States

State 1
 sys.operation = start_op              sys.target.state = start_st           sys.s1.state = start_st
 sys.s2.state = start_st               sys.s3.state = start_st               sys.s4.state = start_st
 ctr.index = 0

State 2
 sys.operation = upload-video          sys.target.state = t1                 sys.s1.state = a1
 sys.s2.state = b1                     sys.s3.state = c1                     sys.s4.state = d1
 ctr.index = 1

State 3
 sys.operation = upload-video          sys.target.state = t1                 sys.s1.state = a1
 sys.s2.state = b1                     sys.s3.state = c1                     sys.s4.state = d1
 ctr.index = 3

State 4
 sys.operation = upload-photo          sys.target.state = t1                 sys.s1.state = a1
 sys.s2.state = b1                     sys.s3.state = c1                     sys.s4.state = d1
 ctr.index = 4

State 5
 sys.operation = put-caption           sys.target.state = t2                 sys.s1.state = a1
 sys.s2.state = b1                     sys.s3.state = c1                     sys.s4.state = d2
 ctr.index = 4

State 6
 sys.operation = translate             sys.target.state = t3                 sys.s1.state = a1
 sys.s2.state = b1                     sys.s3.state = c1                     sys.s4.state = d3
 ctr.index = 4

State 7
 sys.operation = archive               sys.target.state = t4                 sys.s1.state = a1
 sys.s2.state = b1                     sys.s3.state = c1                     sys.s4.state = d1
 ctr.index = 3

State 8
 sys.operation = archive               sys.target.state = t4                 sys.s1.state = a1
 sys.s2.state = b1                     sys.s3.state = c2                     sys.s4.state = d1
 ctr.index = 3

State 9
 sys.operation = archive               sys.target.state = t4                 sys.s1.state = a2
 sys.s2.state = b1                     sys.s3.state = c1                     sys.s4.state = d1
 ctr.index = 1

State 10
 sys.operation = archive               sys.target.state = t4                 sys.s1.state = a2
 sys.s2.state = b1                     sys.s3.state = c1                     sys.s4.state = d1
 ctr.index = 3

State 11
 sys.operation = upload-video          sys.target.state = t1                 sys.s1.state = a2
 sys.s2.state = b1                     sys.s3.state = c1                     sys.s4.state = d1
 ctr.index = 3

State 12
 sys.operation = upload-photo          sys.target.state = t1                 sys.s1.state = a2
 sys.s2.state = b1                     sys.s3.state = c1                     sys.s4.state = d1
 ctr.index = 4

State 13
 sys.operation = put-caption           sys.target.state = t2                 sys.s1.state = a2
 sys.s2.state = b1                     sys.s3.state = c1                     sys.s4.state = d2
 ctr.index = 4

State 14
```

```
 sys.operation = translate            sys.target.state = t3               sys.s1.state = a2
 sys.s2.state = b1                    sys.s3.state = c1                   sys.s4.state = d3
 ctr.index = 4

State 15
 sys.operation = archive              sys.target.state = t4               sys.s1.state = a2
 sys.s2.state = b1                    sys.s3.state = c2                   sys.s4.state = d1
 ctr.index = 3

State 16
 sys.operation = upload-video         sys.target.state = t1               sys.s1.state = a3
 sys.s2.state = b1                    sys.s3.state = c1                   sys.s4.state = d1
 ctr.index = 3

State 17
 sys.operation = upload-photo         sys.target.state = t1               sys.s1.state = a3
 sys.s2.state = b1                    sys.s3.state = c1                   sys.s4.state = d1
 ctr.index = 4

State 18
 sys.operation = put-caption          sys.target.state = t2               sys.s1.state = a3
 sys.s2.state = b1                    sys.s3.state = c1                   sys.s4.state = d2
 ctr.index = 4

State 19
 sys.operation = translate            sys.target.state = t3               sys.s1.state = a3
 sys.s2.state = b1                    sys.s3.state = c1                   sys.s4.state = d3
 ctr.index = 4

State 20
 sys.operation = archive              sys.target.state = t4               sys.s1.state = a3
 sys.s2.state = b1                    sys.s3.state = c1                   sys.s4.state = d1
 ctr.index = 1

State 21
 sys.operation = archive              sys.target.state = t4               sys.s1.state = a3
 sys.s2.state = b1                    sys.s3.state = c1                   sys.s4.state = d1
 ctr.index = 3

State 22
 sys.operation = archive              sys.target.state = t4               sys.s1.state = a3
 sys.s2.state = b1                    sys.s3.state = c2                   sys.s4.state = d1
 ctr.index = 1

State 23
 sys.operation = archive              sys.target.state = t4               sys.s1.state = a3
 sys.s2.state = b1                    sys.s3.state = c2                   sys.s4.state = d1
 ctr.index = 3

State 24
 sys.operation = upload-video         sys.target.state = t1               sys.s1.state = a1
 sys.s2.state = b1                    sys.s3.state = c2                   sys.s4.state = d1
 ctr.index = 1

State 25
 sys.operation = upload-photo         sys.target.state = t1               sys.s1.state = a1
 sys.s2.state = b1                    sys.s3.state = c2                   sys.s4.state = d1
 ctr.index = 4

State 26
 sys.operation = put-caption          sys.target.state = t2               sys.s1.state = a1
 sys.s2.state = b1                    sys.s3.state = c2                   sys.s4.state = d2
 ctr.index = 4

State 27
 sys.operation = translate            sys.target.state = t3               sys.s1.state = a1
 sys.s2.state = b1                    sys.s3.state = c2                   sys.s4.state = d3
 ctr.index = 4

Automaton Transitions
 From 1 to  2 3 4
 From 2 to  9 10
 From 3 to  8
 From 4 to  5
 From 5 to  6
 From 6 to  7
 From 7 to  2 3 4
 From 8 to  2 3 4
 From 9 to  16 17
 From 10 to  11 12
 From 11 to  15
 From 12 to  13
 From 13 to  14
 From 14 to  9 10
 From 15 to  11 12
 From 16 to  22 23
 From 17 to  18
 From 18 to  19
 From 19 to  20 21
 From 20 to  2 3 4
```

```
 From 21 to  16 17
 From 22 to  24 25
 From 23 to  16 17
 From 24 to  15
 From 25 to  26
 From 26 to  27
 From 27 to  8

Automaton has 27 states, and 45 transitions
user time: 0.047 s
BDD nodes allocated: 10363
max amount of BDD nodes allocated: 10363
Bytes allocated: 655424
```

Appendix C

C.1 SMV Code of the Simple Example in Sect. 3.1.1

```
MODULE main
 VAR
   sys: system Sys(ctr.index);
   ctr: system Ctr;
 DEFINE
   good := (ctr.initial & sys.initial) | !(sys.failure);

MODULE Ctr
 VAR
   index : 0..3;
 INIT
   index = 0
 TRANS
   case
    index=0 : next(index)!=0;
    index!=0 : next(index)!=0;
   esac
 DEFINE
   initial := (index=0);

MODULE Sys(index)
 VAR
  operation : {start_op,fill-pump,glucose-test,reset,inject-high-dose,inject-low-dose,
               blood-pressure-test,emergency-call,turn-on-green,turn-on-red,turn-on-yellow};
  target : Target(operation);
  B1 : Behavior1(index,operation);
  B2 : Behavior2(index,operation);
  B3 : Behavior3(index,operation);
 DEFINE
  initial := (B1.initial & B2.initial & B3.initial & target.initial & operation=start_op);
  failure := (B1.failure |B2.failure |B3.failure) |
             (target.final & !(B1.final & B2.final & B3.final));

----------Target behavior----------
MODULE Target(op)
 VAR
  state : {start_st,t2,t3,t4,t5,t6,t7};
 INIT
  state = start_st & op = start_op
 TRANS
   case
     state = start_st & op = start_op : next(state) = t2 & next(op) = glucose-test;
     state = t2  & op = glucose-test  : next(state) = t3 & next(op) in {reset,inject-low-dose,inject-high-dose};
     state = t3  & op = reset  : next(state) = t7 & next(op) in {turn-on-green};
     state = t3  & op = inject-low-dose  : next(state) = t4 & next(op) in {turn-on-yellow};
     state = t3  & op = inject-high-dose : next(state) = t5 & next(op) in {blood-pressure-test};
     state = t7  & op = turn-on-green : next(state) = t2 & next(op) = glucose-test;
     state = t4  & op = turn-on-yellow : next(state) = t2 & next(op) = glucose-test;
     state = t5  & op = blood-pressure-test : next(state) = t6 & next(op) in {turn-on-red,emergency-call};
     state = t6  & op = turn-on-red : next(state) = t2 & next(op) = glucose-test;
     state = t6  & op = emergency-call : next(state) = t2 & next(op) = glucose-test;
   esac

 DEFINE
   initial := state=start_st & op=start_op;
   final := state in {t2};

----------Available behavior #1----------

MODULE Behavior1(index,operation)
 VAR
   state : {start_st,a1,a2};
 INIT
   state=start_st
 TRANS
   case
    state=start_st & operation=start_op  & index=0: next(state)=a1;
    (index != 1) : next(state) = state;
    (state=a1 & operation = fill-pump) : next(state) in {a1};
    (state=a1 & operation = glucose-test) : next(state) in {a2};
    (state=a2 & operation = reset) : next(state) in {a1};
    (state=a2 & operation = inject-high-dose) : next(state) in {a1};
    (state=a2 & operation = inject-low-dose) : next(state) in {a1};
   esac
```

```
 DEFINE
   initial := state=start_st & operation=start_op  & index = 0;
   failure := index = 1 & !((state = a1 & operation in {fill-pump,glucose-test}) |
                     (state = a2 & operation in {inject-high-dose,inject-low-dose,reset}));
   final := state in {a1};

----------End of available behavior #1----------

----------Available behavior #2-----------

MODULE Behavior2(index,operation)
 VAR
   state : {start_st,b1,b2};
 INIT
   state=start_st
 TRANS
   case
    state=start_st & operation=start_op  & index=0: next(state)=b1;
    (index != 2) : next(state) = state;
    (state=b1 & operation = blood-pressure-test) : next(state) in {b2};
    (state=b2 & operation = blood-pressure-test) : next(state) in {b2};
    (state=b2 & operation = reset) : next(state) in {b1};
    (state=b2 & operation = emergency-call) : next(state) in {b1};
   esac
  DEFINE
   initial := state=start_st & operation=start_op  & index = 0;
   failure := index = 2 & !((state = b1 & operation in {blood-pressure-test}) |
                     (state = b2 & operation in {emergency-call,blood-pressure-test,reset}));
   final := state in {b1,b2};

----------End of available behavior #2----------

----------Available behavior #3-----------

MODULE Behavior3(index,operation)
 VAR
   state : {start_st,c1};
 INIT
   state=start_st
 TRANS
   case
    state=start_st & operation=start_op  & index=0: next(state)=c1;
    (index != 3) : next(state) = state;
    (state=c1 & operation = turn-on-green) : next(state) in {c1};
    (state=c1 & operation = turn-on-yellow) : next(state) in {c1};
    (state=c1 & operation = turn-on-red) : next(state) in {c1};
    (state=c1 & operation = emergency-call) : next(state) in {c1};
   esac
  DEFINE
   initial := state=start_st & operation=start_op  & index = 0;
   failure := index = 3 & !((state = c1 & operation in {turn-on-green,turn-on-yellow,
                      turn-on-red,emergency-cal}));
   final := state in {c1};

----------End of available behavior #3----------

---------------TLV output (Controller generator)---------------

Check Realizability
Specification is realizable
Check that a symbolic strategy is correct
Transition relation is complete
All winning states satisfy invariant
Automaton States

State 1
 sys.operation = start_op               sys.target.state = start_st           sys.s1.state = start_st
 sys.s2.state = start_st                sys.s3.state = start_st               ctr.index = 0

State 2
 sys.operation = glucose-test           sys.target.state = t2                 sys.s1.state = a1
 sys.s2.state = b1                      sys.s3.state = c1                     ctr.index = 1

State 3
 sys.operation = inject-high-dose       sys.target.state = t3                 sys.s1.state = a2
 sys.s2.state = b1                      sys.s3.state = c1                     ctr.index = 1

State 4
 sys.operation = reset                  sys.target.state = t3                 sys.s1.state = a2
 sys.s2.state = b1                      sys.s3.state = c1                     ctr.index = 1

State 5
 sys.operation = inject-low-dose        sys.target.state = t3                 sys.s1.state = a2
 sys.s2.state = b1                      sys.s3.state = c1                     ctr.index = 1

State 6
 sys.operation = turn-on-yellow         sys.target.state = t4                 sys.s1.state = a1
 sys.s2.state = b1                      sys.s3.state = c1                     ctr.index = 3

State 7
```

```
 sys.operation = turn-on-green          sys.target.state = t7               sys.s1.state = a1
 sys.s2.state = b1                      sys.s3.state = c1                   ctr.index = 3

State 8
 sys.operation = blood-pressure-test    sys.target.state = t5               sys.s1.state = a1
 sys.s2.state = b1                      sys.s3.state = c1                   ctr.index = 2

State 9
 sys.operation = emergency-call         sys.target.state = t6               sys.s1.state = a1
 sys.s2.state = b2                      sys.s3.state = c1                   ctr.index = 2

State 10
 sys.operation = emergency-call         sys.target.state = t6               sys.s1.state = a1
 sys.s2.state = b2                      sys.s3.state = c1                   ctr.index = 3

State 11
 sys.operation = turn-on-red            sys.target.state = t6               sys.s1.state = a1
 sys.s2.state = b2                      sys.s3.state = c1                   ctr.index = 3

State 12
 sys.operation = glucose-test           sys.target.state = t2               sys.s1.state = a1
 sys.s2.state = b2                      sys.s3.state = c1                   ctr.index = 1

State 13
 sys.operation = inject-high-dose       sys.target.state = t3               sys.s1.state = a2
 sys.s2.state = b2                      sys.s3.state = c1                   ctr.index = 1

State 14
 sys.operation = reset                  sys.target.state = t3               sys.s1.state = a2
 sys.s2.state = b2                      sys.s3.state = c1                   ctr.index = 1

State 15
 sys.operation = inject-low-dose        sys.target.state = t3               sys.s1.state = a2
 sys.s2.state = b2                      sys.s3.state = c1                   ctr.index = 1

State 16
 sys.operation = turn-on-yellow         sys.target.state = t4               sys.s1.state = a1
 sys.s2.state = b2                      sys.s3.state = c1                   ctr.index = 3

State 17
 sys.operation = turn-on-green          sys.target.state = t7               sys.s1.state = a1
 sys.s2.state = b2                      sys.s3.state = c1                   ctr.index = 3

State 18
 sys.operation = blood-pressure-test    sys.target.state = t5               sys.s1.state = a1
 sys.s2.state = b2                      sys.s3.state = c1                   ctr.index = 2

Automaton Transitions

 From 1 to  2
 From 2 to  3  4  5
 From 3 to  8
 From 4 to  7
 From 5 to  6
 From 6 to  2
 From 7 to  2
 From 8 to  9 10 11
 From 9 to  2
 From 10 to  12
 From 11 to  12
 From 12 to  13 14 15
 From 13 to  18
 From 14 to  17
 From 15 to  16
 From 16 to  12
 From 17 to  12
 From 18 to  9 10 11

Automaton has 18 states, and 26 transitions
user time: 0.031 s
BDD nodes allocated: 7368
max amount of BDD nodes allocated: 7368
Bytes allocated: 589888
```

C.2 A Subset of TCTL Formulas in Uppaal

Let us assume that φ and ϕ be the state formulas. The available TCTL formulas in UPPAAL are:

- `A[]`φ
- `E[]`φ
- `A<>`φ
- `E<>`φ
- φ`-->`ϕ
- `A[]not deadlock`
- `E<>deadlock`

It should be noted that a state formula is evaluated for a state. The safety properties can be written as a formula of the form `A[]`φ or `E[]`φ. The former states that φ should be true for all reachable states, while the latter is used to check whether there exists a path such that φ is always true. The liveness properties are expressed by a formula of the form `A<>`φ, meaning that φ is eventually satisfied on all paths. In addition, there is another form (i.e., φ`-->`ϕ) called *response*. This formula states that whenever φ has been satisfied, then the state formula ϕ will eventually be satisfied. A reachability property is written as `E<>`φ and is satisfied if there is a state on a path such that φ is true at that state. Finally, deadlock occurs when a state has no outgoing transition. The formula `E<>deadlock` or `A[]not deadlock` must be used to detect such situations.

Appendix D

D.1 Proof Sketch of Theorem 4.2.2

Basis

Case $p = 1$ in Algorithm 2:

It is evident that $o = 1$, where $o = \mathit{Lift}(R^{\min}, 1, 1)$.

Inductive hypothesis

Assume that the assertion holds for processes p of length smaller than n.

Inductive step

Case $p = p_1 \oplus p_2$ in Algorithm 2, where the length of $p = n$:

Let $o_i = \mathit{Lift}(R^{\min}, p_i, \_)$ for $i = 1, 2$. By the corresponding case in Algorithm 2, let $o = o_1 \oplus o_2$. Notably, the last parameter is the same as the corresponding input parameter. Assume that $\mathcal{TS}_{p_1}$, $\mathcal{TS}_{p_2}$, and $\mathcal{TS}_p$ are the transition systems induced by p_1, p_2 and p, respectively. Likewise, let $\mathcal{TS}_{o_1}$, $\mathcal{TS}_{o_2}$, and $\mathcal{TS}_o$ be the extended transition systems with conditions on their transitions, which are induced by o_1, o_2 and o, respectively.

From the inductive hypothesis, there exists a relation $R_i \subseteq \mathcal{TS}_{p_i} \times \mathcal{TS}_{o_i}$ such that:

$$p_i \leftrightarrows^{I_n}_{R_i} o_i, \quad i = 1, 2. \tag{1}$$

Let $R \subseteq \mathcal{TS}_p \times \mathcal{TS}_o$. The construction of R is as follows:

$$R = R_1 \cup R_2 \cup \{\langle p_1 \oplus p_2, o_1 \oplus o_2\rangle\} \setminus \{\langle p_1, o_1\rangle, \langle p_2, o_2\rangle\}.$$

It must be shown that $p \leftrightarrows^{I_n}_{R} o$.

– The satisfaction of the condition like the second condition of Def. 4.1.16 is immediate: $(p_1 \oplus p_2)\downarrow$ if and only if $(o_1 \oplus o_2)\downarrow$ by Rules R5,[1] since $p_i\downarrow$ if and only if $o_i\downarrow$ by (1), $i = 1, 2$.

1. The rules for "$\oplus$" are similar to "$+$" when the processes are deterministic.

– The satisfaction of conditions like the last two ones of Def. 4.1.16 can be done in one step.

For $i = 1$ or 2 :

$p_i \xrightarrow{a} p_i'$ if and only if $o_i \xrightarrow{\langle a,- \rangle} o_i'$ ($\langle p_i, o_i \rangle \in R_i$ by (1))

$p_1 \oplus p_2 \xrightarrow{a} p_i'$ if and only if $o_1 \oplus o_2 \xrightarrow{\langle a,- \rangle} o_i'$ holds (Rules R5)

$\langle p_1 \oplus p_2, o_1 \oplus o_2 \rangle \in R$ ($\langle p_i', o_i' \rangle \in R_i$ by (1))

The conditions are then verified.

Notably, the index of operation for an orchestrator is not important, since the index-less bisimulation is considered.

Case $p = p_1^*$ in Algorithm 2, where the length of $p = n$:

Let $o_1 = \mathit{Lift}(R^{\min}, p_1, \_)$. By the corresponding case in Algorithm 2, let $o = o_1^*$. Notably, the last parameter is the same as the corresponding input parameter. Assume that $\mathcal{TS}_{p_1}$ and $\mathcal{TS}_p$ are the transition systems induced by p_1 and p, respectively. Likewise, let $\mathcal{TS}_{o_1}$ and $\mathcal{TS}_o$ be the extended transition systems with conditions on their transitions, which are induced by o_1 and o, respectively.

Assume that every state p_1' is renamed by $p_1' \cdot p_1^*$ in the transition graph $\mathcal{TS}_{p_1}$. Likewise, every state o_1' is renamed by $o_1' \cdot o_1^*$ in $\mathcal{TS}_{o_1}$.

From the inductive hypothesis and renaming, there exists a relation $R_1 \subseteq \mathcal{TS}_{p_1} \times \mathcal{TS}_{o_1}$ such that:

$$p_1 \cdot p_1^* \leftrightarrows_{R_1}^{I_n} o_1 \cdot o_1^*. \tag{2}$$

Let $R \subseteq \mathcal{TS}_p \times \mathcal{TS}_o$. The construction of R is as follows:

$$R = R_1 \cup \{\langle p_1^*, o_1^* \rangle\} \setminus \{\langle p_1 \cdot p_1^*, o_1 \cdot o_1^* \rangle\}.$$

It must be shown that $p \leftrightarrows_R^{I_n} o$.

– The satisfaction of the condition like the second condition of Def. 4.1.16 is immediate: $p_1^*\downarrow$ if and only if $o_1^*\downarrow$ by Rules R6.

– The satisfaction of conditions like the last two ones of Def. 4.1.16 can be done

in one step.

$$p_1 \cdot p_1^* \xrightarrow{a} p_1' \cdot p_1^* \text{ if and only if } o_1 \cdot o_1^* \xrightarrow{\langle a,- \rangle} o_1' \cdot o_1^*$$
$$(\langle p_1 \cdot p_1^*, o_1 \cdot o_1^* \rangle \in R_1 \text{ by (2) and renaming})$$
$$p_1^* \xrightarrow{a} p_1' \cdot p_1^* \text{ if and only if } o_1^* \xrightarrow{\langle a,- \rangle} o_1' \cdot o_1^* \text{ holds} \quad \text{(Rules R6)}$$
$$\langle p_1^*, o_1^* \rangle \in R \quad (\langle p_1' \cdot p_1^*, o_1' \cdot o_1^* \rangle \in R_1 \text{ by (2)})$$

The conditions are then verified.

Case $p = p_1 \cdot p_2$ in Algorithm 2, where the length of $p = n$:

Let $o_1 = \mathit{Lift}(R^{\min}, p_1, r)$ and $o_2 = \mathit{Lift}(R^{\min}, p_2, p_2)$. By the corresponding case in Algorithm 2, let $o = o_1 \cdot o_2$. Assume that $\mathcal{TS}_{p_1}$, $\mathcal{TS}_{p_2}$, and $\mathcal{TS}_p$ are the transition systems induced by p_1, p_2 and p, respectively. Likewise, let $\mathcal{TS}_{o_1}$, $\mathcal{TS}_{o_2}$, and $\mathcal{TS}_o$ be the extended transition systems with conditions on their transitions, which are induced by o_1, o_2 and o, respectively.

Assume that every state p_1' is renamed by $p_1' \cdot p_2$ in the transition graph $\mathcal{TS}_{p_1}$. Likewise, every state o_1' is renamed by $o_1' \cdot o_2$ in $\mathcal{TS}_{o_1}$.

From the inductive hypothesis, there exist the relations $R_1 \subseteq \mathcal{TS}_{p_1} \times \mathcal{TS}_{o_1}$ and $R_2 \subseteq \mathcal{TS}_{p_2} \times \mathcal{TS}_{o_2}$ such that:

$$p_1 \leftrightarrows^{I_n}_{R_1} o_1 \text{ and } p_2 \leftrightarrows^{I_n}_{R_2} o_2. \tag{3}$$

Let $R \subseteq \mathcal{TS}_p \times \mathcal{TS}_o$. The construction of R is as follows:

$$R = R_1 \cup R_2.$$

It must be shown that $p \leftrightarrows^{I_n}_{R} o$.

- The satisfaction of the condition like the second condition of Def. 4.1.16 is immediate: $(p_1 \cdot p_2)\downarrow$ if and only if $(o_1 \cdot o_2)\downarrow$ by Rule R10, since $p_1\downarrow$ if and only if $o_1\downarrow$ and $p_2\downarrow$ if and only if $o_2\downarrow$ by (3).
- The satisfaction of conditions like the last two ones of Def. 4.1.16 can be done in one step. Since a move in p_1 or p_2 has two different semantic rules (see Rules R4), the both cases should be proven separately.

1. Case $p_1 \xrightarrow{a} p_1'$:

$$p_1 \xrightarrow{a} p_1' \text{ if and only if } o_1 \xrightarrow{\langle a,-\rangle} o_1' \qquad (\langle p_1, o_1\rangle \in R_1 \text{ by (3)})$$

$$p_1 \cdot p_2 \xrightarrow{a} p_1' \cdot p_2 \text{ if and only if } o_1 \cdot o_2 \xrightarrow{\langle a,-\rangle} o_1' \cdot o_2 \text{ holds}$$

(Rules R4 and renaming)

$$\langle p_1 \cdot p_2, o_1 \cdot o_2\rangle \in R \qquad (\langle p_1', o_1'\rangle \in R_1 \text{ by (3) and renaming})$$

2. Case $p_1\!\downarrow$ and $p_2 \xrightarrow{a} p_2'$:

$$p_1\!\downarrow \text{ and } p_2 \xrightarrow{a} p_2' \text{ if and only if } o_1\!\downarrow \text{ and } o_2 \xrightarrow{\langle a,-\rangle} o_2'$$

$$(\langle p_1, o_1\rangle \in R_1 \text{ and } \langle p_2, o_2\rangle \in R_2 \text{ by (3)})$$

$$p_1 \cdot p_2 \xrightarrow{a} p_2' \text{ if and only if } o_1 \cdot o_2 \xrightarrow{\langle a,-\rangle} o_2' \text{ holds} \qquad \text{(Rules R4)}$$

$$\langle p_1 \cdot p_2, o_1 \cdot o_2\rangle \in R \qquad (\langle p_2', o_2'\rangle \in R_2 \text{ by (3)})$$

Given the satisfaction of conditions in both cases, the last two conditions of Def. 4.1.16 are then verified.

Case $p = a.p'$ in Algorithm 2, where the length of $p = n$:

Let $o_i' = \mathit{Lift}(R_i^{\min}, p', r')$. By the corresponding case in Algorithm 2, let $o = \vee_i\, [q_i\colon \langle a, k_i\rangle].o_i'$. Assume that $\mathcal{TS}_{p'}$ and $\mathcal{TS}_p$ are the transition systems induced by p' and p. Likewise, let $\mathcal{TS}_{o_i'}$ and $\mathcal{TS}_o$ be the extended transition systems with conditions on their transitions, which are induced by o_i' and o, respectively, for i such that $\langle r, q_i\rangle \in R$.

From the inductive hypothesis, there exist a relation $R_i' \subseteq \mathcal{TS}_{p'} \times \mathcal{TS}_{o_i'}$ such that:

$$p' \leftrightarrows^{I_n}_{R'} o_i'. \tag{4}$$

Let $R \subseteq \mathcal{TS}_p \times \mathcal{TS}_o$. The construction of R is as follows:

$$R = \cup_i R_i' \cup \{\langle a.p', o\rangle \mid \langle r, q_i\rangle \in R^{\min}\}.$$

It must be shown that $p \leftrightarrows^{I_n}_{R} o$.

– The second condition of Def. 4.1.16 is not applicable, since both $a.p'$ and $\langle a, -\rangle.o_i'$ cannot be final states.

– The satisfaction of conditions like the last two ones of Def. 4.1.16 can be done in one step.

$a.p \xrightarrow{a} p'$ if and only if $c_i : \langle a, - \rangle.o'_i$ holds for all i (Rule R3, Rule R13)

$\langle a.p, o \rangle \in R$ ($\langle p', o'_i \rangle \in R'_i$ for all i)

The conditions are then verified.

D.2 Proof Sketch of Lemmas in Sect. 4.2.2

Lemma 4.2.1. There exists a relation $R' \subseteq \mathcal{TS}_o \times \mathcal{TS}_s$ such that $o \preceq_{R'} s$.

Proof.

$t \leftrightarroweq^{I_n}_{R} o$, where $R \subseteq \mathcal{TS}_t \times \mathcal{TS}_o$ is an index-less bisimulation relation;
(Assumption and Theorem 4.2.2)

$o \preceq^{I_n}_{R} t$; (Def. 4.2.6)

$o \preceq_{R'} s$, where $R' \subseteq \mathcal{TS}_o \times \mathcal{TS}_s, R' = R \circ R^{\min}$.
(Assumption, $R^{\min}$ is minimal, and Remark 4.2.5)

□

Lemma 4.2.2. There exists a relation $R' \subseteq \mathcal{TS}_o \times \mathcal{TS}_{t \rfloor o \lfloor s}$ such that $o \leftrightarroweq^{I_n}_{R'} t \rfloor o \lfloor s$.

Proof.

$q\downarrow$ if and only if $p \rfloor q \lfloor r\downarrow$ holds; (Rules R14, Theorem 4.2.2, and Lemma 4.2.1)

$q \xrightarrow{\langle a,k,r \rangle} q'$ if and only if $p \rfloor q \lfloor r \xrightarrow{a} p' \rfloor q' \lfloor r'$ holds;
(Rules R14, Theorem 4.2.2, and Lemma 4.2.1)

$o \leftrightarroweq^{I_n}_{R'} t \rfloor o \lfloor s$, where $R' \subseteq \mathcal{TS}_o \times \mathcal{TS}_{t \rfloor o \lfloor s}$ is an index-less bisimulation relation.
(Def. 4.2.6)

□

Lemma 4.2.3. There exists a relation $R' \subseteq \mathcal{TS}_{t \rfloor o \lfloor s} \times \mathcal{TS}_s$ such that $t \rfloor o \lfloor s \preceq^{I_n}_{R'} s$.

Proof.

$t \rfloor o \lfloor s \leftrightarroweq^{I_n}_{R_1} o$, where $R_1 \subseteq \mathcal{TS}_{t \rfloor o \lfloor s} \times \mathcal{TS}_o$ is an index-less bisimulation relation;
(Lemma 4.2.2)

$o \preceq_{R_2} s$, where $R_2 \subseteq \mathcal{TS}_o \times \mathcal{TS}_s$ is a simulation relation; (Lemma 4.2.1)

$t \rfloor o \lfloor s \preceq^{I_n}_{R'} s$, where $R' \subseteq \mathcal{TS}_{t \rfloor o \lfloor s} \times \mathcal{TS}_s, R' = R_1 \circ R_2$. (Remark 4.2.2)

□

Lemma 4.2.4. There exists a relation $R' \subseteq \mathcal{TS}_o \times \mathcal{TS}_{t\rfloor o}$ such that $o \leftrightarrows^{I_n}_{R'} t\rfloor o$.

Proof.

$q\downarrow$ if and only if $p\rfloor q\downarrow$ holds; (Rules R15 and Theorem 4.2.2)

$q \xrightarrow{\langle a,-,-\rangle} q'$ if and only if $p\rfloor q \xrightarrow{a} p'\rfloor q'$ holds; (Rules R15 and Theorem 4.2.2)

$o \leftrightarrows^{I_n}_{R'} t\rfloor o$, where $R' \subseteq \mathcal{TS}_o \times \mathcal{TS}_{t\rfloor o}$ is an index-less bisimulation relation. (Def. 4.2.6)

□

Lemma 4.2.5. There exists a relation $R' \subseteq \mathcal{TS}_o \times \mathcal{TS}_{o\lfloor s}$ such that $o \leftrightarrows^{I_n}_{R'} o\lfloor s$.

Proof.

$q\downarrow$ if and only if $q\lfloor r\downarrow$ holds; (Rules R16 and Lemma 4.2.1)

$q \xrightarrow{\langle a,k,r\rangle} q'$ if and only if $q\lfloor r \xrightarrow{a} q'\lfloor r'$ holds; (Rules R16 and Lemma 4.2.1)

$o \leftrightarrows^{I_n}_{R'} o\lfloor s$, where $R' \subseteq \mathcal{TS}_o \times \mathcal{TS}_{o\lfloor s}$ is an index-less bisimulation relation. (Def. 4.2.6)

□

Lemma 4.2.6. There exists a relation $R' \subseteq \mathcal{TS}_{t\rfloor o} \times \mathcal{TS}_{o\lfloor s}$ such that $t\rfloor o \leftrightarrows_{R'} o\lfloor s$.

Proof.

$t\rfloor o \leftrightarrows^{I_n}_{R_1} o$, where $R_1 \subseteq \mathcal{TS}_{t\rfloor o} \times \mathcal{TS}_o$ is an index-less bisimulation relation; (Lemma 4.2.4)

$o \leftrightarrows^{I_n}_{R_2} o\lfloor s$, where $R_2 \subseteq \mathcal{TS}_o \times \mathcal{TS}_{o\lfloor s}$ is an index-less bisimulation relation; (Lemma 4.2.5)

$t\rfloor o \leftrightarrows_{R'} o\lfloor s$, where $R' \subseteq \mathcal{TS}_{t\rfloor o} \times \mathcal{TS}_{o\lfloor s}$, $R' = R_1 \circ R_2$. (Remark 4.2.1 and $\alpha(t\rfloor o) = \alpha(o\lfloor s)$)

□

Lemma 4.2.7. There exists a relation $R' \subseteq \mathcal{TS}_t \times \mathcal{TS}_{t\rfloor o}$ such that $t \leftrightarrows_{R'} t\rfloor o$.

Proof.

$t \leftrightarrows^{I_n}_{R_1} o$, where $R_1 \subseteq \mathcal{TS}_t \times \mathcal{TS}_o$ is an index-less bisimulation relation; (Theorem 4.2.2)

$o \leftrightarrows^{I_n}_{R_2} t\rfloor o$, where $R_2 \subseteq \mathcal{TS}_o \times \mathcal{TS}_{t\rfloor o}$ is an index-less bisimulation relation; (Lemma 4.2.4)

$t \leftrightarrows_{R'} t\rfloor o$, where $R' \subseteq \mathcal{TS}_t \times \mathcal{TS}_{t\rfloor o}, R' = R_1 \circ R_2$. (Remark 4.2.1 and $\alpha(t) = \alpha(t\rfloor o)$)

□

Lemma 4.2.8. There exists a relation $R' \subseteq \mathcal{TS}_t \times \mathcal{TS}_{t\rfloor o\lfloor s}$ such that $t \leftrightarrows_{R'} t\rfloor o\lfloor s$.

Proof.

$t \leftrightarrows^{I_n}_{R_1} o$, where $R_1 \subseteq \mathcal{TS}_t \times \mathcal{TS}_o$ is an index-less bisimulation relation; (Theorem 4.2.2)

$o \leftrightarrows^{I_n}_{R_2} t\rfloor o\lfloor s$, where $R_2 \subseteq \mathcal{TS}_o \times \mathcal{TS}_{t\rfloor o\lfloor s}$ is an index-less bisimulation relation; (Lemma 4.2.2)

$t \leftrightarrows_{R'} t\rfloor o\lfloor s$, where $R' \subseteq \mathcal{TS}_t \times \mathcal{TS}_{t\rfloor o\lfloor s}, R' = R_1 \circ R_2$. (Remark 4.2.1 and $\alpha(t) = \alpha(t\rfloor o\lfloor s)$)

□

Lemma 4.2.9. There exists a relation $R' \subseteq \mathcal{TS}_{o\lfloor s} \times \mathcal{TS}_s$ such that $o\lfloor s \preceq^{I_n}_{R'} s$.

Proof.

$o \leftrightarrows^{I_n}_{R_1} o\lfloor s$, where $R_1 \subseteq \mathcal{TS}_o \times \mathcal{TS}_{o\lfloor s}$ is an index-less bisimulation relation; (Lemma 4.2.5)

$o \preceq_{R_2} s$, where $R_2 \subseteq \mathcal{TS}_o \times \mathcal{TS}_s$ is a simulation relation; (Lemma 4.2.1)

$o\lfloor s \preceq^{I_n}_{R'} s$, where $R' \subseteq \mathcal{TS}_{o\lfloor s} \times \mathcal{TS}_s, R' = R_1 \circ R_2$. (Remark 4.1.3)

□

Lemma 4.2.10. There exists a relation $R' \subseteq \mathcal{TS}_{t\rfloor o} \times \mathcal{TS}_{(t\rfloor o)|(o\lfloor s)}$ such that $t\rfloor o \leftrightarrows_{R'} (t\rfloor o)|(o\lfloor s)$ and there exists a relation $R'' \subseteq \mathcal{TS}_{o\lfloor s} \times \mathcal{TS}_{(t\rfloor o)|(o\lfloor s)}$ such that $o\lfloor s \leftrightarrows_{R''} (t\rfloor o)|(o\lfloor s)$.

Proof.

Case R':

$p\rfloor q\downarrow$ if and only if $(p\rfloor q)|(q\lfloor r)\downarrow$ holds; (Rule R10 and Lemma 4.2.6)

$p\rfloor q \xrightarrow{a} q'\lfloor r'$ if and only if $(p\rfloor q)|(q\lfloor r) \xrightarrow{a} (p'\rfloor q')|(q'\lfloor r')$ holds; (Rule R8 and Lemma 4.2.6)

$t\rfloor o \leftrightarrows_{R'} (t\rfloor o)|(o\lfloor s)$, where $R' \subseteq \mathcal{TS}_{t\rfloor o} \times \mathcal{TS}_{(t\rfloor o)|(o\lfloor s)}$ is a bisimulation relation.

Case R'':

$q\lfloor r\downarrow$ if and only if $(p\rfloor q)|(q\lfloor r)\downarrow$ holds; (Rule R10 and Lemma 4.2.6)

$q\lfloor r \xrightarrow{a} q'\lfloor r'$ if and only if $(p\rfloor q)|(q\lfloor r) \xrightarrow{a} (p'\rfloor q')|(q'\lfloor r')$ holds; (Rule R8 and Lemma 4.2.6)

$o\lfloor s \leftrightarrows_{R''} (t\rfloor o)|(o\lfloor s)$, where $R'' \subseteq \mathcal{TS}_{o\lfloor s} \times \mathcal{TS}_{(t\rfloor o)|(o\lfloor s)}$ is a bisimulation relation.

□

D.3 Proof Sketch of Proposition 4.2.1

Although two paths are possible to reach $(t \rfloor o)|(o \lfloor s)$ from $t \rfloor o \lfloor s$ in Fig 4.3, one of them is considered for the proof as follows.

$o \leftrightarrows^{I_n}_{R_1} t \rfloor o$, where $R_1 \subseteq \mathcal{TS}_o \times \mathcal{TS}_{t \rfloor o}$ is an index-less bisimulation relation; (Lemma 4.2.4)

$t \rfloor o \leftrightarrows_{R_2} (t \rfloor o)|(o \lfloor s)$, where $R_2 \subseteq \mathcal{TS}_{t \rfloor o} \times \mathcal{TS}_{(t \rfloor o)|(o \lfloor s)}$ is a bisimulation relation; (Lemma 4.2.10)

$o \leftrightarrows^{I_n}_{R_3} (t \rfloor o)|(o \lfloor s)$, where $R_3 \subseteq \mathcal{TS}_o \times \mathcal{TS}_{(t \rfloor o)|(o \lfloor s)}, R_3 = R_1 \circ R_2$; (Remark 4.2.4)

$t \rfloor o \lfloor s \leftrightarrows^{I_n}_{R_4} o$, where $R_4 \subseteq \mathcal{TS}_{t \rfloor o \lfloor s} \times \mathcal{TS}_o$ is an index-less bisimulation relation; (Lemma 4.2.2)

$t \rfloor o \lfloor s \leftrightarrows_R (t \rfloor o)|(o \lfloor s)$, where $R \subseteq \mathcal{TS}_{t \rfloor o \lfloor s} \times \mathcal{TS}_{(t \rfloor o)|(o \lfloor s)}, R = R_4 \circ R_3$. (Remark 4.2.1 and $\alpha(t \rfloor o \lfloor s) = \alpha((t \rfloor o)|(o \lfloor s)))$

□

D.4 Proof Sketch of an Extension in Algorithm 2

Inductive hypothesis

Assume that the assertion holds for processes p of length smaller than n.

Inductive step

Case $p = p_1|[H]|p_2$ as an extension of Algorithm 2, where the length of $p = n$:

Let $o_i = \mathit{Lift}(R^{\min}, p_i, \_)$ for $i = 1, 2$. By the corresponding case in Algorithm 2, let $o = o_1|[H]|o_2$, where o is deadlock free based on the condition provided in Sect. 4.3.2. Notably, the last parameter is the same as the corresponding input parameter. Assume that $\mathcal{TS}_{p_1}$, $\mathcal{TS}_{p_2}$, and $\mathcal{TS}_p$ are the transition systems induced by p_1, p_2 and p, respectively. Likewise, let $\mathcal{TS}_{o_1}$, $\mathcal{TS}_{o_2}$, and $\mathcal{TS}_o$ be the extended transition systems with conditions on their transitions, which are induced by o_1, o_2 and o, respectively.

From the inductive hypothesis, there exists a relation $R_i \subseteq \mathcal{TS}_{p_i} \times \mathcal{TS}_{o_i}$ such that:

$$p_i \leftrightarrows^{I_n}_{R_i} o_i, \quad i = 1, 2. \tag{1}$$

Let $R \subseteq (\mathcal{TS}_{p_1} \times \mathcal{TS}_{p_2}) \times (\mathcal{TS}_{o_1} \times \mathcal{TS}_{o_2})$.
It must be shown that $p \leftrightarrows^{I_n}_{R} o$.

– The satisfaction of the condition like the second condition of Def. 4.1.16 is immediate: $(p_1|[H]|p_2)\downarrow$ if and only if $(o_1|[H]|o_2)\downarrow$ by the assumptions that $(p_1|[H]|p_2)$ and $(o_1|[H]|o_2)$ are deadlock free (see the condition in Sect. 4.3.2) and the inductive hypothesis $p_i\downarrow$ if and only if $o_i\downarrow$ by (1), $i = 1, 2$.

– The satisfaction of conditions like the last two ones of Def. 4.1.16 can be done in one step.

$$\begin{aligned}
&\text{For } a \notin H : p_i \xrightarrow{a} p_i' \text{ if and only if } o_i \xrightarrow{\langle a,- \rangle} o_i' && (\langle p_i, o_i\rangle \in R_i \text{ by (1)})\\
&\qquad p_1|[H]|p_2 \xrightarrow{a} p_i' \text{ if and only if } o_1|[H]|o_2 \xrightarrow{\langle a,- \rangle} o_i' \text{ holds} && \text{(Rules R9)}\\
&\qquad \langle\langle p_1, p_2\rangle, \langle o_1, o_2\rangle\rangle \in R && (\langle p_i', o_i'\rangle \in R_i \text{ by (1)})
\end{aligned}$$

$$\begin{aligned}
&\text{For } a \in H : p_i \xrightarrow{a} p_i' \text{ if and only if } o_i \xrightarrow{\langle a,- \rangle} o_i' && (\langle p_i, o_i\rangle \in R_i \text{ by (1)})\\
&\qquad p_1|[H]|p_2 \xrightarrow{a} p_1'|[H]|p_2' \text{ if and only if } o_1|[H]|o_2 \xrightarrow{\langle a,- \rangle} o_1'|[H]|o_2' \text{ holds} && \text{(Rules R9)}\\
&\qquad \langle\langle p_1, p_2\rangle, \langle o_1, o_2\rangle\rangle \in R && (\langle p_i', o_i'\rangle \in R_i \text{ by (1)})
\end{aligned}$$

The conditions are then verified.

Notably, the index of operation for an orchestrator is not important, since the index-less bisimulation is considered. □

D.5 Proof Sketch of Lemma 4.4.1

Case $t_1 \cdot t_2$:
Determinism—When $t_1\downarrow$, this means that a subprocess 1 has been chosen among the operations involved in the choice (this set, denoted $\alpha_{\downarrow}(t_1)$ for this choice, can be empty). By hypothesis, $\alpha_{\downarrow}(t_1) \cap \alpha_1(t_2) = \emptyset$. So, $t_1 \cdot t_2$ is deterministic.
Termination—Since t_1 and t_2 are well-formed, $t_1\downarrow$ and $t_2\downarrow$. By applying Rule R10, $(t_1 \cdot t_2)\downarrow$.

Case $t_1 \oplus t_2$:
Determinism—The processes t_1 and t_2 are well-formed, in particular they are deterministic. By hypothesis, $\alpha_1(t_1) \cap \alpha_1(t_2) = \emptyset$. So, $t_1 \oplus t_2$ is deterministic.
Termination—Since t_1 and t_2 are well-formed, $t_1\!\downarrow$ and $t_2\!\downarrow$. By applying Rule R5, $(t_1 \oplus t_2)\!\downarrow$.
Case t^*:
Determinism—Since t is well-formed, t is deterministic. So, t^* is deterministic.
Termination—This is immediate by applying Rule R6.
Case $t_1|[H]|t_2$:
Determinism—The processes t_1 and t_2 are well-formed, in particular they are deterministic. Since t_1 and t_2 synchronize on common actions, $t_1|[H]|t_2$ is deterministic by Rule R9.
Termination—Since t_1 and t_2 are well-formed, $t_1\!\downarrow$ and $t_2\!\downarrow$. By hypothesis $t_1|[H]|t_2$ is deadlock free. So, $(t_1|[H]|t_2)\!\downarrow$. □

D.6 Proof Sketch of Proposition 4.4.1

Let $R^{\min} = R_1^{\min} \cup R_2^{\min}$. Then,

$$\begin{aligned} o &= \mathit{Lift}(R_1^{\min} \cup R_2^{\min}, t_1 \oplus t_2, t_1 \oplus t_2) \\ &= \mathit{Lift}(R_1^{\min} \cup R_2^{\min}, t_1, t_1) \oplus \mathit{Lift}(R_1^{\min} \cup R_2^{\min}, t_2, t_2) \\ &= \mathit{Lift}(R_1^{\min}, t_1, t_1) \oplus \mathit{Lift}(R_2^{\min}, t_2, t_2) \\ &= o_1 \oplus o_2 \end{aligned}$$

□

D.7 Proof Sketch of Proposition 4.4.2

Let $R'^{\min} = R^{\min}$. Then,

$$\begin{aligned} o' &= \mathit{Lift}(R^{\min}, t^*, t^*) \\ &= \mathit{Lift}(R^{\min}, t, t)^* \\ &= o^* \end{aligned}$$

□

D.8 Proof Sketch of Proposition 4.4.3

Let $R^{\min} = R_1^{\min} \cup R_2^{\min}$. Then,

$$\begin{aligned} o &= \mathit{Lift}(R_1^{\min} \cup R_2^{\min}, t_1 \cdot t_2, t_1 \cdot t_2) \\ &= \mathit{Lift}(R_1^{\min} \cup R_2^{\min}, t_1, t_1) \cdot \mathit{Lift}(R_1^{\min} \cup R_2^{\min}, t_2, t_2) \\ &= \mathit{Lift}(R_1^{\min}, t_1, t_1) \cdot \mathit{Lift}(R_2^{\min}, t_2, t_2) \\ &= o_1 \cdot o_2 \end{aligned}$$

□

D.9 Proof Sketch of Proposition 4.4.4

Let $R^{\min} = \{\langle\langle p_1, p_2\rangle, r\rangle \mid \langle p_1, r\rangle \in R_1^{\min} \text{ and } \langle p_2, r\rangle \in R_2^{\min}\}$. Then,

$$\begin{aligned} o &= \mathit{Lift}(R^{\min}, t_1|[H]|t_2, t_1|[H]|t_2) \\ &= \mathit{Lift}(R^{\min}, t_1, t_1)|[H]|\mathit{Lift}(R^{\min}, t_2, t_2) \\ &= o_1|[H]|o_2 \end{aligned}$$

Note that, when the orchestrators o_1 and o_2 synchronize on the operation a, they also synchronize on a state of the system (see Rules R21). □

Appendix E

E.1 SMV Code and TLV Output of Example 5.1.1

```
MODULE main
 VAR
   sys: system Sys(ctr.index);
   ctr: system Ctr;
 DEFINE
   good := (ctr.initial & sys.initial) | !(sys.failure);

MODULE Ctr
 VAR
   index : 0..3;
 INIT
   index = 0
 TRANS
   case
    index=0 : next(index)!=0;
    index!=0 : next(index)!=0;
   esac
 DEFINE
   initial := (index=0);

MODULE Sys(index)
 VAR
  resource : {start_op, StorageSpace205GB, StorageSpace210GB, StorageSpace220GB, Windows7, Windows8,
              WindowsVista, SQL-Server2005, SQL-Server2008};
  threshold : 0..10;
  target : Target(resource,threshold);
  B1 : Behavior1(index,resource,threshold);
  B2 : Behavior2(index,resource,threshold);
  B3 : Behavior3(index,resource,threshold);
 DEFINE
  initial := (B1.initial & B2.initial & B3.initial & target.initial & resource=start_op & threshold = 0);
  failure := (B1.failure |B2.failure |B3.failure) |
             (target.final & !(B1.final & B2.final & B3.final));

----------Target behavior----------

MODULE Target(resource,threshold)
 VAR
   state : {start_st,t1,t2,t3};
 INIT
   state = start_st & resource = start_op & threshold = 0
 TRANS
  case
    state = start_st & resource = start_op & threshold = 0 : next(state) = t1 &
           next(resource) in {StorageSpace210GB} & next(threshold) = 3;
    state = t1 & resource in {StorageSpace210GB} & threshold = 3 : next(state) = t2 &
           next(resource) in {Windows7} & next(threshold) = 8;
    state = t2 & resource in {Windows7} & threshold = 8 : next(state) = t3 &
           next(resource) in {SQL-Server2008} & next(threshold) = 7;
    state = t3 & in {SQL-Server2008} & threshold = 7 : next(state) = t1 &
           next(resource) in {StorageSpace210GB} & next(threshold) = 3;
   esac
  DEFINE
   initial := state = start_st & resource = start_op & threshold = 0;
   final := state in {t1};

----------Available behavior #1----------
```

```
MODULE Behavior1(index,resource,threshold)
 VAR
   state : {start_st,a1,a2};
 INIT
   state = start_st
 TRANS
  case
   state = start_st & resource = start_op & threshold = 0 & index = 0 : next(state) in {a1};
   (index != 1) : next(state) = state;
   (state=a1 & resource in {Windows8,Windows7} & threshold<=8) : next(state) in {a2};
   (state=a2 & resource in {SQL-Server2005,SQL-Server2008} & threshold<=8) : next(state) in {a1};
  esac
 DEFINE
  initial := state = start_st & resource = start_op & threshold = 0 & index = 0;
  failure := index = 1 & !((state = a1 & resource in {Windows8,Windows7} & threshold<=8) |
             (state = a2 & resource in {SQL-Server2005,SQL-Server2008} & threshold<=8));
  final := state in {a1};

----------End of available behavior #1----------

----------Available behavior #2-----------

MODULE Behavior2(index,resource,threshold)
 VAR
   state : {start_st,b1,b2};
 INIT
   state = start_st
 TRANS
  case
   state = start_st & resource = start_op & threshold = 0 & index = 0 : next(state) in {b1};
   (index != 2) : next(state) = state;
   (state=b1 & resource in {StorageSpace210GB,StorageSpace220GB} & threshold<=3) : next(state) in {b2};
   (state=b2 & resource in {WindowsVista,Windows7} & threshold<=8) : next(state) in {b1};
   (state=b1 & resource in {SQL-Server2005,SQL-Server2008} & threshold<=8) : next(state) in {b1};
  esac
 DEFINE
  initial := state = start_st & resource = start_op & threshold = 0 & index = 0;
  failure := index = 2 & !((state = b1 & resource in {StorageSpace210GB,StorageSpace220GB} & threshold<=3) |
             (state = b1 & resource in {SQL-Server2005,SQL-Server2008} & threshold<=8) |
             (state = b2 & resource in {WindowsVista,Windows7} & threshold<=8));
  final := state in {b1};

----------End of available behavior #2----------

----------Available behavior #3-----------

MODULE Behavior3(index,resource,threshold)
 VAR
   state : {start_st,c1,c2};
 INIT
   state = start_st
 TRANS
  case
   state = start_st & resource = start_op & threshold = 0 & index = 0 : next(state) in {c1};
   (index != 3) : next(state) = state;
   (state=c1 & resource in {SQL-Server2008} & threshold==10) : next(state) in {c2};
   (state=c2 & resource in {SQL-Server2008} & threshold==10) : next(state) in {c1};
   (state=b1 & resource in {StorageSpace205GB,StorageSpace210GB} & threshold<=3) : next(state) in {b1};
  esac
 DEFINE
  initial := state = start_st & resource = start_op & threshold = 0 & index = 0;
  failure := index = 3 & !((state = c1 & resource in {SQL-Server2008} & threshold==10) |
             (state = c1 & resource in {StorageSpace205GB,StorageSpace210GB} & threshold<=3) |
             (state = c2 & resource in {SQL-Server2008} & threshold==10));
  final := state in {c1};

----------End of available behavior #3----------
```

```
---------------TLV output (Controller generator)---------------

Check Realizability
Specification is realizable
Check that a symbolic strategy is correct
Transition relation is complete
All winning states satisfy invariant
Automaton States

State 1
 sys.resource = start_op            sys.threshold = 0              sys.target.state = start_st
 sys.B1.state = start_st            sys.B2.state = start_st        sys.B3.state = start_st
 ctr.index = 0

State 2
 sys.resource = Storage210GB        sys.threshold = 3              sys.target.state = t1
 sys.B1.state = a1                  sys.B2.state = b1              sys.B3.state = c1
 ctr.index = 2

State 3
 sys.resource = Storage210GB        sys.threshold = 3              sys.target.state = t1
 sys.B1.state = a1                  sys.B2.state = b1              sys.B3.state = c1
 ctr.index = 3

State 4
 sys.resource = Windows7            sys.threshold = 8              sys.target.state = t2
 sys.B1.state = a1                  sys.B2.state = b1              sys.B3.state = c1
 ctr.index = 1

State 5
 sys.resource = SQL_Server2008      sys.threshold = 7              sys.target.state = t3
 sys.B1.state = a2                  sys.B2.state = b1              sys.B3.state = c1
 ctr.index = 1

State 6
 sys.resource = Windows7            sys.threshold = 8              sys.target.state = t2
 sys.B1.state = a1                  sys.B2.state = b2              sys.B3.state = c1
 ctr.index = 2

State 7
 sys.resource = SQL_Server2008      sys.threshold = 7              sys.target.state = t3
 sys.B1.state = a1                  sys.B2.state = b1              sys.B3.state = c1
 ctr.index = 2

Automaton Transitions

 From 1 to  2  3
 From 2 to  6
 From 3 to  4
 From 4 to  5
 From 5 to  2  3
 From 6 to  7
 From 7 to  2  3

Automaton has 7 states, and 10 transitions
user time: 0.015 s
BDD nodes allocated: 7955
max amount of BDD nodes allocated: 7955
Bytes allocated: 589888
```

E.2 SMV Code and TLV Output of Example 5.2.1

```
MODULE main
 VAR
   sys: system Sys(ctr.index);
   ctr: system Ctr;
 DEFINE
   good := (ctr.initial & sys.initial) | !(sys.failure);

MODULE Ctr
 VAR
   index : 0..3;
 INIT
   index = 0
 TRANS
   case
    index=0 : next(index)!=0;
    index!=0 : next(index)!=0;
   esac
 DEFINE
   initial := (index=0);

MODULE Sys(index)
 VAR
  service : {start_se,Hotel,Meals,Bus,Airplaneticket};
  Attr_values : {start_pr,star3,star4,star2,vegetable,meat,economy,first,business,shuttle,tour};
  target : Target(service,Attr_values);
  B1 : Behavior1(index,service,Attr_values);
  B2 : Behavior2(index,service,Attr_values);
  B3 : Behavior3(index,service,Attr_values);
 DEFINE
  initial := (B1.initial & B2.initial & B3.initial & target.initial & service=start_se & Attr_values=start_pr);
  failure := (B1.failure | B2.failure | B3.failure) |
               (target.final & !(B1.final & B2.final & B3.final));


----------Target behavior----------

MODULE Target(service,preference)
 VAR
  state : {start_st,t1,t2,t3,t4};
  Fav_hotel : {star4};
  Alt_hotel : {star3};
  Dis_air : {first};
  Fav_meals : {vegetable};
  Fav_bus : {shuttle};
 INIT
  state = start_st & service = start_se & preference=start_pr
 TRANS
  case
   state = start_st & service = start_se & preference=start_pr: next(state) = t1 &
           next(service) in {Airplaneticket} & ((next(preference) in {economy,business,first})
           & (next(preference) notin Dis_air));
   state = t1 & service = Airplaneticket & ((preference in {economy,business,first}) &
           (preference notin Dis_air)) : next(state) in {t2} & next(service) in {Hotel}
           & next(preference) in (Fav_hotel union Alt_hotel);
   state = t2 & service = Hotel & preference in (Fav_hotel union Alt_hotel): next(state) = t3
           & next(service) in {Meals} & next(preference) in Fav_meals;
   state = t3 & service = Meals &  preference in Fav_meals : next(state) = t4 & next(service)
           in {Bus} & next(preference) in Fav_bus;
   state = t4 & service = Bus &  preference in Fav_bus : next(state) = t1 & next(service)
           in {Airplaneticket} & ((next(preference) in {economy,business,first}) &
           (next(preference) notin Dis_air));
  esac
 DEFINE
  initial := state=start_st & service=start_se & preference=start_pr;
```

```
  final := state in {t1}; -- final state(s)

----------Available behavior #1-----------

MODULE Behavior1(index,service,Attr_values)
 VAR
  state : {start_st,a1,a2,a3};
 INIT
  state=start_st
 TRANS
  case
   state=start_st & service=start_se & Attr_values=start_pr & index=0: next(state)=a1;
   (index != 1) : next(state) = state;
   state=a1 & (service in {Hotel} & Attr_values in {star3}) : next(state) in {a1,a2};
   state=a2 & (service in {Meals} &  Attr_values in {vegetable,meat}): next(state) in {a3};
   state=a3 & (service in {Airplaneticket} & Attr_values in {economy,bussiness}): next(state) in {a1};
  esac
 DEFINE
  initial := state=start_st & service=start_se & index = 0 & Attr_values=start_pr;
  failure := index = 1 & !((state = a1 & (service in {Hotel} & Attr_values in {star3}))|
             (state = a2 & (service in {Meals} & Attr_values in {vegetable,meat}))|(state = a3
             & (service in {Airplaneticket} & Attr_values in {economy,business})));
  final := state in {a1,a2};

----------End of available behavior #1----------

----------Available behavior #2-----------

MODULE Behavior2(index,service,Attr_values)
 VAR
  state : {start_st,b1,b2};
 INIT
  state=start_st
 TRANS
  case
   state=start_st & service=start_se & Attr_values=start_pr & index=0: next(state)=b1;
   (index != 2) : next(state) = state;
   state=b1 & (service in {Meals} & Attr_values in {vegetable}) : next(state) in {b1};
   state=b1 & (service in {Bus} &  Attr_values in {tour}): next(state) in {b2};
   state=b2 & (service in {Airplaneticket} & Attr_values in {economy}): next(state) in {b1};
  esac
 DEFINE
  initial := state=start_st & service=start_se & index = 0 & Attr_values=start_pr;
  failure := index = 2 & !((state = b1 & (service in {Meals} & Attr_values in {vegetable}))|
             (state = b1 & (service in {Bus} & Attr_values in {tour}))|(state = b2
             & (service in {Airplaneticket} & Attr_values in {economy})));
  final := state in {b1};

----------End of available behavior #2----------

----------Available behavior #3-----------

MODULE Behavior3(index,service,Attr_values)
 VAR
  state : {start_st,c1,c2};
 INIT
  state=start_st
 TRANS
  case
   state=start_st & service=start_se & Attr_values=start_pr & index=0: next(state)=c1;
   (index != 3) : next(state) = state;
   state=c1 & (service in {Airplaneticket} & Attr_values in {economy,business,first}) : next(state) in {c2};
   state=c2 & (service in {Hotel} &  Attr_values in {star2,star3,star4}): next(state) in {c2};
   state=c2 & (service in {Bus} & Attr_values in {shuttle}): next(state) in {c1};
  esac
 DEFINE
  initial := state=start_st & service=start_se & index = 0 & Attr_values=start_pr;
  failure := index = 3 & !((state = c1 & (service in {Airplaneticket} & Attr_values in {economy,business,first}))|
```

```
            (state = c2 & (service in {Hotel} & Attr_values in {star2,star3,star4}))|(state = c2
            & (service in {Bus} & Attr_values in {shuttle})));
  final := state in {c1};

----------End of available behavior #3----------

---------------TLV output (Controller generator)---------------

Check Realizability
Specification is realizable
Check that a symbolic strategy is correct
Transition relation is complete
All winning states satisfy invariant
Automaton States

State 1
 sys.service = start_se                 sys.Attr_values = start_pr         sys.target.state = start_st
 sys.B1.state = start_st                sys.B2.state = start_st            sys.B3.state = start_st
 ctr.index = 0

State 2
 sys.service = Airplaneticket           sys.Attr_values = business         sys.target.state = t1
 sys.B1.state = a1                      sys.B2.state = b1                  sys.B3.state = c1
 ctr.index = 3

State 3
 sys.service = Airplaneticket           sys.Attr_values = economy          sys.target.state = t1
 sys.B1.state = a1                      sys.B2.state = b1                  sys.B3.state = c1
 ctr.index = 3

State 4
 sys.service = Hotel                    sys.Attr_values = star4            sys.target.state = t2
 sys.B1.state = a1                      sys.B2.state = b1                  sys.B3.state = c2
 ctr.index = 3

State 5
 sys.service = Hotel                    sys.Attr_values = star3            sys.target.state = t2
 sys.B1.state = a1                      sys.B2.state = b1                  sys.B3.state = c2
 ctr.index = 1

State 6
 sys.service = Hotel                    sys.Attr_values = star3            sys.target.state = t2
 sys.B1.state = a1                      sys.B2.state = b1                  sys.B3.state = c2
 ctr.index = 3

State 7
 sys.service = Meals                    sys.Attr_values = vegetable        sys.target.state = t3
 sys.B1.state = a1                      sys.B2.state = b1                  sys.B3.state = c2
 ctr.index = 2

State 8
 sys.service = Bus                      sys.Attr_values = shuttle          sys.target.state = t4
 sys.B1.state = a1                      sys.B2.state = b1                  sys.B3.state = c2
 ctr.index = 3

State 9
 sys.service = Meals                    sys.Attr_values = vegetable        sys.target.state = t3
 sys.B1.state = a2                      sys.B2.state = b1                  sys.B3.state = c2
 ctr.index = 2

State 10
 sys.service = Bus                      sys.Attr_values = shuttle          sys.target.state = t4
 sys.B1.state = a2                      sys.B2.state = b1                  sys.B3.state = c2
 ctr.index = 3

State 11
 sys.service = Airplaneticket           sys.Attr_values = business         sys.target.state = t1
 sys.B1.state = a2                      sys.B2.state = b1                  sys.B3.state = c1
```

```
 ctr.index = 3

State 12
 sys.service = Airplaneticket           sys.Attr_values = economy         sys.target.state = t1
 sys.B1.state = a2                      sys.B2.state = b1                 sys.B3.state = c1
 ctr.index = 3

State 13
 sys.service = Hotel                    sys.Attr_values = star4           sys.target.state = t2
 sys.B1.state = a2                      sys.B2.state = b1                 sys.B3.state = c2
 ctr.index = 3

State 14
 sys.service = Hotel                    sys.Attr_values = star3           sys.target.state = t2
 sys.B1.state = a2                      sys.B2.state = b1                 sys.B3.state = c2
 ctr.index = 3


Automaton Transitions

 From 1 to  2 3
 From 2 to  4 5 6
 From 3 to  4 5 6
 From 4 to  7
 From 5 to  7 9
 From 6 to  7
 From 7 to  8
 From 8 to  2 3
 From 9 to  10
 From 10 to  11 12
 From 11 to  13 14
 From 12 to  13 14
 From 13 to  9
 From 14 to  9

Automaton has 14 states, and 24 transitions
user time: 0.016 s
BDD nodes allocated: 9393
max amount of BDD nodes allocated: 9393
Bytes allocated: 655424
```

Appendix F

F.1 Implementation of Example 6.1.1 in PuLP

```
# List of agents
agents = ['a1', 'a2', 'a3', 'a4', 'a5']
# Costs assigned to agents
costs = {'a1': 8.0, 'a2': 3.0, 'a3': 6.0, 'a4': 10.0, 'a5': 6.0}
# Task 2 of the goal
task2 = {'a1': 1, 'a2': 1, 'a3': 0, 'a4': 1, 'a5': 0}
# Task 3 of the goal
task3 = {'a1': 0, 'a2': 1, 'a3': 0, 'a4': 1, 'a5': 0}
# Task 5 of the goal
task5 = {'a1': 1, 'a2': 0, 'a3': 0, 'a4': 1, 'a5': 1}
# Degree of robustness
K = 0
while True:
  # Create the problem
  problem = LpProblem("Example 1.1 of MATES 2017", LpMinimize)
  # Create dictionary 'A' for the variables 'x' associated with agents
  x = LpVariable.dicts("A", agents, lowBound = 0, upBound = 1, cat = LpInteger)
  # The objective function
  problem += lpSum([costs[i]*x[i] for i in agents]), "total cost"
  # Block of constraints (one per task in the goal)
  problem += lpSum([task2[i] * x[i] for i in agents]) >= K+1, "task 2"
  problem += lpSum([task3[i] * x[i] for i in agents]) >= K+1, "task 3"
  problem += lpSum([task5[i] * x[i] for i in agents]) >= K+1, "task 5"
  problem.solve()
  # Display the status of the solution
  print "Status:", LpStatus[problem.status]
  if problem.status == LpStatusOptimal:
     # Display the optimal value
     print "Cost of the team = ", value(problem.objective)
     # Display the optimal solution
     for v in problem.variables():
         print v.name, "=", v.varValue
  else:
     # Write the problem data to an .lp file
     problem.writeLP("teamMATES2017Ex1.lp")
     break
  K = K+1

---------------PuLP output---------------

Status: Optimal
Cost of the team =  9.0
A_a1 = 0.0
A_a2 = 1.0
A_a3 = 0.0
A_a4 = 0.0
A_a5 = 1.0
Status: Optimal
Cost of the team =  19.0
A_a1 = 0.0
A_a2 = 1.0
A_a3 = 0.0
A_a4 = 1.0
A_a5 = 1.0
Status: Infeasible
```

F.2 Implementation of Example 6.3.2 in PuLP

```
# List of agents
agents = ['b1','b2', 'b3']
# List of compositions with rank 1 or 2
compositions = ['c1', 'c2', 'c3','c4', 'c6']
# List of links between a composition and an agent
links = ['a11', 'a12', 'a13', 'a21', 'a22', 'a23',
         'a31', 'a32', 'a33', 'a41', 'a42', 'a43',
         'a61', 'a62', 'a63']
# Costs assigned to compositions
kappa1 = {'c1': 1.0, 'c2': 1.0, 'c3': 1.0, 'c4': 1.0, 'c6': 1.0}
# Costs assigned to links
kappa3 = {'a11': 1.0, 'a12': 1.0, 'a13': 1.0,
          'a21': 1.0, 'a22': 1.0, 'a23': 1.0,
          'a31': 1.0, 'a32': 1.0, 'a33': 1.0,
          'a41': 1.0, 'a42': 1.0, 'a43': 1.0,
          'a61': 1.0, 'a62': 1.0, 'a63': 1.0}
# Number of agents per compositions
nbAperC = {'c1': 2, 'c2': 2, 'c3': 2, 'c4': 2, 'c6': 3}
# Agents in each composition
In = {'a11': 0, 'a12': 1, 'a13': 1,
      'a21': 0, 'a22': 1, 'a23': 1,
      'a31': 0, 'a32': 1, 'a33': 1,
      'a41': 0, 'a42': 1, 'a43': 1,
      'a61': 1, 'a62': 1, 'a63': 1}
# Task  'airplaneticket'
task1 = {'b1': 0, 'b2': 0, 'b3': 1}
# Task  'hotel'
task2 = {'b1': 1, 'b2': 0, 'b3': 1}
# Task 'meals'
task3 = {'b1': 0, 'b2': 1, 'b3': 0}
# Task 'bus'
task4 = {'b1': 0, 'b2': 0, 'b3': 1}
# Degree of robustness
K = 0
while True:
  # Create the problem
  problem = LpProblem("Example 2.1 of MATES 2017", LpMinimize)
  # Create dictionary 'A' for the variables 'x_j' associated with agents
  x = LpVariable.dicts("A", agents,
                       lowBound = 0, upBound = 1, cat = LpInteger)
  # Create dictionary 'C' for the variables 'y_i' associated with compositions
  y = LpVariable.dicts("C", compositions,
                       lowBound = 0, upBound = 1, cat = LpInteger)
  # Create dictionary 'L' for the variables 'x_ij' associated with links
  xy = LpVariable.dicts("L", links,
                        lowBound = 0, upBound = 1, cat = LpInteger)
  # The objective function
  problem += lpSum([kappa1[i]*y[i] for i in compositions] +
                   [kappa2[j]*x[j] for j in agents] +
                   [kappa3[k]*xy[k] for k in links]), "total cost
                   # Block of constraints (6)
  problem += LpAffineExpression([(xy['a11'],1), (xy['a21'],1), (xy['a31'],1),
               (xy['a41'],1), (x['b1'],-1)]) >= 0, "for agent 1"
  problem += LpAffineExpression([(xy['a12'],1), (xy['a22'],1), (xy['a32'],1),
               (xy['a42'],1), (x['b2'],-1)]) >= 0, "for agent 2"
  problem += LpAffineExpression([(xy['a13'],1), (xy['a23'],1), (xy['a33'],1),
               (xy['a43'],1), (x['b3'],-1)]) >= 0, "for agent 3"
  # Block of constraints (7)
  problem += LpAffineExpression([(xy['a11'],In['a11']),
               (xy['a12'],In['a12']), (xy['a13'],In['a13']),
               (y['c1'],-nbAperC['c1'])]) == 0, "for composition 1"
  problem += LpAffineExpression([(xy['a21'],In['a21']),
               (xy['a22'],In['a22']), (xy['a23'],In['a23']),
               (y['c2'],-nbAperC['c2'])]) == 0, "for composition 2"
```

```
  problem += LpAffineExpression([(xy['a31'],In['a31']),
              (xy['a32'],In['a32']), (xy['a33'],In['a33']),
              (y['c3'],-nbAperC['c3'])]) == 0, "for composition 3"
  problem += LpAffineExpression([(xy['a41'],In['a41']),
              (xy['a42'],In['a42']), (xy['a43'],In['a43']),
              (y['c4'],-nbAperC['c4'])]) == 0, "for composition 4"
  problem += LpAffineExpression([(xy['a61'],In['a61']),
              (xy['a62'],In['a62']), (xy['a63'],In['a63']),
              (y['c6'],-nbAperC['c6'])]) == 0, "for composition 6"
  # Block of constraints (8)
  problem += lpSum([task1[j] * x[j] for j in agents]) >= K+1, "task 1"
  problem += lpSum([task2[j] * x[j] for j in agents]) >= K+1, "task 2"
  problem += lpSum([task3[j] * x[j] for j in agents]) >= K+1, "task 3"
  problem += lpSum([task4[j] * x[j] for j in agents]) >= K+1, "task 4"
  problem.solve()
  # Display the status of the solution
  print "Status:", LpStatus[problem.status]
  if problem.status == LpStatusOptimal:
     # Display the optimal value
     print "Cost of the team = ", value(problem.objective)
     # Display the optimal solution
     for v in problem.variables():
         print v.name, "=", v.varValue
  else:
     # Write the problem data to an .lp file
     problem.writeLP("teamMATES2017Ex2.lp")
     break
  K = K+1

---------------PuLP output---------------

Status: Optimal
Cost of the team =  5.0
A_b1 = 0.0
A_b2 = 1.0
A_b3 = 1.0
C_c1 = 0.0
C_c2 = 0.0
C_c3 = 1.0
C_c4 = 0.0
C_c6 = 0.0
L_a11 = 0.0
L_a12 = 0.0
L_a13 = 0.0
L_a21 = 0.0
L_a22 = 0.0
L_a23 = 0.0
L_a31 = 0.0
L_a32 = 1.0
L_a33 = 1.0
L_a41 = 0.0
L_a42 = 0.0
L_a43 = 0.0
L_a61 = 0.0
L_a62 = 0.0
L_a63 = 0.0
Status: Infeasible
```

F.3 Ruby Program Computing the Number of Compositions

```
require 'set'

print "Enter the number of tasks: "
p = STDIN.gets.strip.to_i

tasks = Array.new(p) { |i| "t" + (i+1).to_s }
Tasks = Set.new(tasks)
print "Tasks : ";   print tasks;   puts
                 # print Tasks.inspect;   puts

def exclusive_selection(s, tasks)
    set_tasks = Set.new(tasks)
  # print s.inspect;   print set_tasks.inspect; print  (s & set_tasks).empty?;  puts
    return (s & set_tasks).empty?
end

def check_for_all_tasks(s)
    if s.size == Tasks.size then return true else return false end
end

agent_tasks = Hash.new
m = 1

print "Combinations of tasks : ";   puts
n = 0
(1..p).each do |k|
  c_p_k = tasks.combination(k).to_a;   n = n + c_p_k.size
  c_p_k.each { |c| agent_tasks[m] = c;   m += 1 }
  print c_p_k.size;   print  ": ";   print c_p_k;   puts
end

print "Number of agents: ";   puts n
agent_tasks.each do |key, value| print "a" + key.to_s;   print " : "
                                print value;   puts end

print "Selection of compositions... ";   puts
l = 0;   m = 0
compositions = Array.new(n) { |i| i+1 }
(1..n).each do |k|
  c_n_k = compositions.combination(k).to_a;   l = l + c_n_k.size
  c_n_k.each { |c| # print c;
               s = Set.new;
               c.each {  |e| s.merge(agent_tasks[e])
                         # remove the following comment signs for the first strategy
                         #  |e| if exclusive_selection(s, agent_tasks[e]) then
                         #         s.merge(agent_tasks[e])
                         #      else
                         #         s.clear; break
                         #     end
                         # print agents[e];  print s.inspect;  puts
                       }
               if check_for_all_tasks(s) then
                  m += 1;   print "--> (";   print m;   print ")"; print c;   puts end }

  # print c_n_k.size;   print  ": ";   print c_n_k;   puts
end

print "Number of potential compositions: ";   puts l
print "Number of selected compositions: ";   puts m
```

Bibliography

[1] A. Aijaz, M. Dohler, A. H. Aghvami, V. Friderikos, and M. Frodigh. Realizing the tactile Internet: haptic communications over next generation 5G cellular networks. *IEEE Wireless Communications*, 24(2):82–89, 2017.

[2] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[3] T. Andreasen, H. Bulskov, and R. Knappe. From ontology over similarity to query evaluation. In R. Bernardi and M. Moortgat, editors, *2nd CoLogNET-ElsNET Symposium - Questions and Answers: Theoretical and Applied Perspectives*, pages 39–50, Amsterdam, The Netherlands, 2003.

[4] R. Arp, B. Smith, and A. D. Spear. *Building Ontologies with Basic Formal Ontology*. MIT Press, Cambridge, Massachusetts, 2015.

[5] L. Atzori, A. Iera, and G. Morabito. The Internet of things: a survey. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 54(15):2787–2805, 2010.

[6] J. C. M. Baeten, T. Basten, and M. A. Reniers. *Process Algebra: Equational Theories of Communicating Processes*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, UK, 2010.

[7] J. C. M. Baeten, D. A. van Beek, B. Luttik, J. Markovski, and J. E. Rooda. A process-theoretic approach to supervisory control theory. In *Proceedings of the 2011 American Control Conference*, pages 4496–4501, San Francisco, CA, 2011.

[8] M. Barati and R. St-Denis. An architecture for semantic service discovery and realizability in cloud computing. In *Proceedings of the 2015 6th International Conference on Network of the Future*, 6 pages, Montreal, Canada, 2015.

[9] M. Barati and R. St-Denis. Behavior composition meets supervisory control. In *Proceedings of the 2015 IEEE International Conference On Systems, Man, and Cybernetics*, pages 115–120, Hong Kong, 2015.

[10] M. Barati and R. St-Denis. Optimal control in a value-based automatic behavior composition framework. In *Proceedings of the 7th International Conference on*

Ultra Modern Telecommunications and Control Systems, pages 1–7, Brno, Czech Republic, 2015.

[11] M. Barati and R. St-Denis. A semantic-based flexible framework for automatic behavior composition. In *Proceedings of the 6th International Conference on Internet Technologies and Applications*, pages 115–120, Wrexham, United Kingdom, 2015.

[12] M. Barati and R. St-Denis. Team formation through preference-based behavior composition. In J. O. Berndt, P. Petta, and R. Unland, editors, *Multiagent System Technologies, 15th German Conference, MATES 2017*, volume 10413 of *Lecture Notes in Artificial Intelligence*, pages 54–71, Leipzig, Germany, 2017.

[13] G. Behrmann, A. David, and K. G. Larsen. A tutorial on Uppaal. In M. Bernardo and F. Corradini, editors, *Formal Methods for the Design of Real-Time Systems*, volume 3826 of *Lecture Notes in Computer Science*, pages 200–236, Berlin, Heidelberg, 2004.

[14] D. Berardi, F. Cheikh, G. De Giacomo, and F. Patrizi. Automatic service composition via simulation. *International Journal of Foundations of Computer Science*, 19(2):429–451, 2008.

[15] J. A. Bergstra and C. A. Middelburg. Thread algebra for strategic interleaving. *Formal Aspects of Computing*, 19(4):445–474, 2007.

[16] P. Bertoli, M. Pistore, and P. Traverso. Automated composition of web services via planning in asynchronous domains. *Artificial Intelligence*, 174(3–4):316–361, 2010.

[17] C. Binder. Preference and similarity between alternatives. *Rationality, Markets, and Morals*, 5(88):120–132, 2014.

[18] M. Bordignon, J. Rashid, M. Broxvall, and A. Saffiotti. Seamless integration of robots and tiny embedded devices in a PEIS-ecology. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3101–3106, San Diego, CA, 2007.

[19] H. Bowman and R. Gómez. How to stop time stopping. *Formal Aspects of Computing*, 18(4):459–493, 2006.

[20] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer, New York, NY, 2008.

[21] D. Côté, M. Embe Jiague, and R. St-Denis. Systems-theoretic view of component-based software development. In L. S. Barbosa and M. Lumpe, editors, *Formal Aspects of Component Software, FACS 2010*, volume 6921 of *Lecture Notes in Computer Science*, pages 163–181, Guimarães, Portugal, 2012.

[22] D. Côté and R. St-Denis. Component-based method for the modeling and control of modular production systems. *IEEE Transactions on Control Systems Technology*, 21(5):1570–1585, 2013.

[23] B. Councill and G. T. Heineman. *Definition of a Software Component and Its Elements*. Addison-Wesley Longman Publishing Co., Boston, MA, 2001.

[24] C. Crawford, Z. Rahaman, and S. Sen. Evaluating the efficiency of robust team formation algorithms. In N. Osman and C. Sierra, editors, *Autonomous Agents and Multiagent Systems*, volume 10002 of *Lecture Notes in Artificial Intelligence*, pages 14–29, Gewerbestrassen, Switzerland, 2016.

[25] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[26] M. Derakhshanmanesh. *Model-Integrating Software Components: Engineering Flexible Software Systems*. Springer Vieweg, Wiesbaden, Germany, 2015.

[27] J. E. Doran, S. Fraiklin, N. R. Jennings, and T. J. Norman. On cooperation in multi-agent systems. *The Knowledge Engineering Review*, 12(3):309–314, 1997.

[28] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina. *Present and Ulterior Software Engineering*, chapter Microservices: Yesterday, Today, and Tomorrow, pages 195–216. Springer, 2017. B. Meyer and M. Mazzara (Eds).

[29] C. Emig, K. Krutz, S. Link, C. Momm, and S. Abeck. Model-driven development of SOA services. Technical report, Universität Karlsruhe (TH), Karlsruhe, Germany, 2007.

[30] J. M. García, D. Ruiz, and A. Ruiz-Cortés. An intuitive and formal description of preferences for semantic web service discovery and ranking. Technical report, Universidad de Sevilla, Sevilla, Spain, 2012.

[31] G. De Giacomo, M. Mecella, and F. Patrizi. *Web Service Foundations*, chapter Automatic service composition based on behaviors: the Roman model, pages 189–214. Springer, New York, 2014. A. Bouguettaya, Q. Z. Sheng, and F. Daniel (Eds).

[32] G. De Giacomo, F. Patrizi, and S. Sardina. Automatic behavior composition synthesis. *Artificial Intelligence*, 196:106–142, 2013.

[33] M. L. Hale, M. T. Gamble, and R. F. Gamble. A design and verification framework for service composition in the cloud. In *Proceedings of the 2013 IEEE 9th World Congress on Services*, pages 317–324, Santa Clara, CA, 2013.

[34] L. Han and B. Dave. Semantic-supported and agent-based decentralized grid resource discovery. *Future Generation Computer Systems*, 24(8):806–812, 2008.

[35] Q.-S. Hua, Y. Wang, D. Yu, and F. C. M. Lau. Dynamic programming based algorithms for set multicover and multiset multicover problems. *Theoretical Computer Science*, 411(26–28):2467–2474, 2010.

[36] A. Jula, E. Sundararajan, and Z. Othman. Cloud computing service composition: a systematic literature review. *Expert Systems with Applications*, 41(8):3809–3824, 2014.

[37] I. Kholod, I. Petuhov, and M. Efimova. Data mining for the Internet of things with fog nodes. In O. Galinina, S. Balandin, and Y. Koucheryavy, editors, *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*, volume 9870 of *Lecture Notes in Computer Science*, pages 25–36, Berlin Heidelberg, 2016.

[38] W. Kießling. Foundations of preferences in database systems. In *Proceedings of the 28th International Conference on Very Large Data Bases*, pages 311–322, Hong Kong, 2002.

[39] B. Konieczek, M. Rethfeldt, F. Golatowski, and D. Timmermann. Real-time communication for the Internet of things using jCoAP. In *Proceedings of IEEE*

18th International Symposium on Real-Time Distributed Computing, pages 134–141, Aucklang, New Zealand, 2015.

[40] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Journal of Software Tools for Technology Transfer*, 1(1):134–152, 1997.

[41] L. Leite, C. E. Moreira, D. Cordeiro, M. A. Gerosa, and F. Kon. Deploying large-scale service compositions on the cloud with the CHOReOS enactment engine. In *Proceedings of the 2014 IEEE 13th International Symposium on Network Computing and Applications*, pages 121–128, Cambridge, MA, 2014.

[42] L. Li, Z. Jin, G. Li, L. Zheng, and Q. Wei. Modeling and analyzing the reliability and cost of service composition in the IoT: a probabilistic approach. In *Proceedings of IEEE 19th International Conference on Web Services*, pages 584–591, Honolulu, HI, 2012.

[43] S. Li, L. D. Xu, and S. Zhao. The Internet of things: a survey. *Information Systems Frontiers*, 17(2):243–259, 2015.

[44] Y. Lustig and M. Y. Vardi. Synthesis from component libraries. *International Journal of Software Tools Technology Transfer*, 15(5–6):603–618, 2013.

[45] D. Martin, M. Burstein, D. McDermott, S. McIlraith, M. Paolucci, K. Sycara, D. L. McGuinness, E. Sirin, and N. Srinivasan. Bringing semantics to web services with OWL-S. *World Wide Web Journal*, 10(3):243–277, 2007.

[46] F. D. O. Militão. Design and implementation of a behaviorally typed programming system for web services. Master's thesis, Departamento de Informática, Universidade Nova de Lisboa, 2008.

[47] T. Okimoto, N. Schwind, M. Clement, T. Ribeiro, K. Inoue, and P. Marquis. How to form a task-oriented robust team. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 395–403, Istanbul, Turkey, 2015.

[48] J.-H. Park, L. Q. Zhe, and S.-D. Kim. Reliable services composition method for the Internet of thing using directed service-object graph deployment scheme. In *Proceedings of 2nd International Conference on Information Science and Security*, pages 1–4, Seoul, South Korea, 2015.

[49] G. Pirró. A semantic similarity metric combining features and intrinsic information content. *Data & Knowledge Engineering*, 68(11):1289–1308, 2009.

[50] A. Pnueli and E. Shahar. A platform for combining deductive with algorithmic verification. In *Proceedings of the International Conference Computer Aided Verification (CAV)*, pages 184–195, New Brunswick, NJ, 1996.

[51] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of IEEE*, 77(1):81–98, 1989.

[52] G. Randelli, L. Marchetti, F. A. Marino, and L. Iocchi. Multi-agent behavior composition through adaptable software architectures and tangible interfaces. In J. Ruiz del Solor, E. Chown, and P. G. Plöger, editors, *RoboCup 2010: Robot Soccer World Cup XIV*, volume 6556 of *Lecture Notes in Computer Science*, pages 278–290, Singapore, 2011.

[53] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke. Middleware for Internet of things: a survey. *IEEE Internet of Things Journal*, 3(1):70–95, 2016.

[54] M. A. Rodriguez and M. J. Egenhofer. Determining semantic similarity among entity classes from different ontologies. *IEEE Transactions on Knowledge and Data Engineering*, 15(2):442–456, 2003.

[55] S. Sardina and G. De Giacomo. Realizing multiple autonomous agents through scheduling of shared devices. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling*, pages 304–312, Sydney, Australia, 2008.

[56] S. Schneider. *Concurrent and Real-time Systems: the CSP Approach*. John Wiley & Sons, Inc., New York, NY, 1999.

[57] K. M. Sim. Agent-based cloud computing. *IEEE Transactions on Services Computing*, 5(4):564–577, 2012.

[58] P. Spiess, S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L. M. Sá de Souza, and V. Trifa. SOA-based integration of the Internet of things in enterprise services. In *Proceedings of IEEE International Conference on Web Services*, pages 968–975, Los Angeles, CA, 2009.

[59] M. H. Ter Beek, A. Bucchiarone, and S. Gnesi. Formal methods for service composition. *Annals of Mathematics, Computing & Teleinformatics*, 1(5):1–10, 2007.

[60] The Economist Intelligence Unit. The Internet of things business index 2017: Transformation in motion. *Report sponsored by ARM and IBM*, 2017.

[61] O. Vermesan, P. Friess, P. Guillemin, H. Sundmaeker, M. Eisenhauer, K. Moessner, F. L. Gall, and P. Cousin. *Internet of Things - Converging Technologies for Smart Environments and Integrated Ecosystems*, chapter Internet of Things Strategic Research and Innovation Agenda, pages 7–152. River Publishers, 2014. O. Vermesan and P. Friess (Eds).

[62] M. Ylianttila, J. Riekki, J. Zhou, K. Athukorala, and E. Gilman. Cloud architecture for dynamic service composition. *International Journal of Grid and High Performance Computing*, 4(2):17–31, 2012.

[63] L. Yongxiang, Y. Xifan, X. Xiangmin, and J. Hong. Formal verification of cloud manufacturing service composition and BPEL codes generation based on extended process calculus. *Information Technology Journal*, 13(11):1779–1785, 2014.