Automating Verification of Non-functional Properties

Chris Chilton, Lukas Holik, Paola Inverardi, Bengt Jonsson, Marta Kwiatkowska, Hongyang Qu, Massimo Tivoli, Xu Wang

▶ **To cite this version:**

Chris Chilton, Lukas Holik, Paola Inverardi, Bengt Jonsson, Marta Kwiatkowska, et al.. Automating Verification of Non-functional Properties. [Research Report] 2012. hal-00805608

HAL Id: hal-00805608
https://inria.hal.science/hal-00805608

Submitted on 28 Mar 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

CONNECT

Emergent Connectors for

Eternal Software Intensive Networked Systems

ICT FET IP Project

Deliverable D2.4

Automating Verification of Non-functional Properties

SEVENTH FRAMEWORK PROGRAMME

http://www.connect-forever.eu

Inria · DOCOMO Euro-Labs · Ambientic · Lancaster University · Thales · Università dell'Aquila · technische universität dortmund · University of Oxford · Uppsala Universitet

Project Number	:	231167
Project Title	:	CONNECT – Emergent Connectors for Eternal Software Intensive Networked Systems
Deliverable Type	:	Report

Deliverable Number	:	D2.4
Title of Deliverable	:	Automating Verification of Non-functional Properties
Nature of Deliverable	:	R
Dissemination Level	:	Public
Internal Version Number	:	1.5
Contractual Delivery Date	:	01 December 2012
Actual Delivery Date	:	01 December 2012
Contributing WPs	:	WP2
Editor(s)	:	Massimo Tivoli (UNIVAQ)
Author(s)	:	Chris Chilton (UOXF), Lukáš Holík (UU), Paola Inverardi (UNIVAQ), Bengt Jonsson (UU), Marta Kwiatkowska (UOXF), Hongyang Qu (UOXF), Massimo Tivoli (UNIVAQ), Xu Wang (UOXF)
Reviewer(s)	:	Bernhard Steffen (TUDO)

Abstract

In this deliverable, we document the progress of WP2 during the fourth year of the CONNECT *project. Following the last reviews' recommendations, the work is organized into two main streams. One stream concerns a quantitative extension of the compositional specification theory devised during the previous two years, while the other integrates the WP2* CONNECT*or algebra with the specification theory so as to support WP3* CONNECT*or synthesis. In particular, the assume-guarantee reasoning framework developed for the compositional specification theory ensures that a* CONNECT*ed system preserves global safety properties by just checking the local properties of its constituent components. The proposed quantitative extension of the compositional specification theory allows the modeling of the real-time performance of networked systems, in addition to enabling the synthesis of* CONNECT*ors that are compatible with both the functional behavior and timing constraints of their environments. Finally, in order to integrate the* CONNECT*or algebra with the specification theory so as to support WP3 synthesis, we defined a method for the automated synthesis of modular* CONNECT*ors. We prove that the behavior of such a* CONNECT*or is equivalent to the behaviour of a monolithic WP3* CONNECT*or. All of the work is evaluated through an application to relevant* CONNECT *scenarios, e.g., the GMES (Global Monitoring for Environment and Security) scenario.*

Keyword List

Specification theory, quotient, synthesis, assume-guarantee, data, quantitative verification, functional and non-functional requirements, mediators, compositional connectors, mediation patterns, interface automata.

Document History

Version	Type of Change	Author(s)
1.0	initial version	Massimo Tivoli (UNIVAQ)
1.1	writing of Section 2.3	Massimo Tivoli (UNIVAQ)
1.2	writing of Section 2.1	UOXF
1.3	writing of Section 2.2	UOXF
1.4	finalization for the internal revision	Massimo Tivoli (UNIVAQ)
1.5	final CONNECT-official version	Massimo Tivoli (UNIVAQ) and UOXF

Table of Contents

List of Figures

1 Introduction

The CONNECT project aims to develop a revolutionary approach to the seamless networking of digital systems by synthesizing, on-the-fly, the CONNECTors via which heterogeneous Networked Systems (NSs) communicate. The role of Work Package 2 (WP2) is to investigate the foundations and verification methods for composable CONNECTors, so that support is provided for composition of NSs, whilst enabling automated learning, reasoning and synthesis.

This document provides an overview of the work undertaken by WP2 in the fourth year of the project. The work is organized into two streams: (i) a *quantitative extension of the compositional specification theory* defined during the third year [12], with an *application to a* CONNECT *case study*, and (ii) the *integration of the connector algebra, defined during the second year [11], with the specification theory and its support for WP3 synthesis*. These two work streams were explicitly mentioned in the reviewers' recommendations from the last review (see Section 1.3), hence we considered them as a priority for this deliverable. Furthermore, the first work stream addresses the main topics of D2.4, as planned in the CONNECT DoW, i.e., "*Investigate a quantitative assume-guarantee framework*" and "*Develop scalable techniques for timed systems based on approximation and abstraction*".

In this chapter, we introduce the two mentioned work streams. Section 1.1 recalls the role of WP2 within the CONNECT project by highlighting the WP2 tasks and objectives. Then, in Section 1.2, we describe how the work undertaken during the fourth year fits the WP2 tasks/objectives. Section 1.3 highlights how this work addresses the last reviewers' recommendations, while Section 1.4 provides an outline for the remainder of the deliverable.

1.1 The role of work package WP2

We recall that the role of WP2 is to investigate the foundations and verification methods for composable connectors, so that support is provided for composition of NSs, whilst enabling automated learning, reasoning and synthesis. The expected outcomes are formalisms, methods and software tools that can be used for the specification, design and development of connectors, allowing for both functional and non-functional properties to be expressed and verified. WP2 thus provides the theoretical underpinning for the work carried out in the other work packages, in the sense that connectors specified in WP2 can be instantiated in WP3 (synthesis), WP4 (learning) and WP5 (dependability analysis).

The remit of WP2 is to develop compositional specification and verification techniques to the extent that they can be successfully applied to the modeling and reasoning of connector behaviours in a compositional manner. To achieve this goal, WP2 is structured into the four tasks:

- **Task 2.1**. Capturing functional and non-functional connector behaviours. This task aims to guide the project by formalizing the notions of connector and component, characterizing the types of interaction and identifying a verification approach, capable of capturing non-functional properties.
- **Task 2.2**. Compositional connector operators. The main thrust here is to formulate a compositional modeling and reasoning framework for components and connectors.
- **Task 2.3**. Rephrasing interoperability in terms of connector behaviours. The aim is to formulate techniques for interoperability checking, in the presence of dynamic behaviours and non-functional properties.
- **Task 2.4**. Reasoning toolset. The focus here is on a quantitative verification framework for connectors and components, capable of handling dynamic scenarios and non-functional properties, which includes algorithms and prototype implementations.

1.2 WP2 progress during the fourth year

Work in the fourth year has focused on completing the remaining objectives of Tasks 2.3 and 2.4, in addition to addressing the recommendations from the third review. Consequently, we have organized this work into two main streams.

One stream of work concerns the integration of the defined *Compositional Specification Theory* [12] with the devised *Assume-Guarantee-based Quantitative Verification Framework* [12], and an application of it to a CONNECT case study. As suggested by the reviewers, to address this issue, we have been working on (i) an assume-guarantee framework for reasoning compositionally about the safety properties satisfied by specifications of components; (ii) the formulation of a quantitative extension to the specification theory; and (iii) the identification of a suitable CONNECT scenario to demonstrate the overall applicability of our verification framework to the objectives of CONNECT. We briefly remark on each strand below:

- The work of (i) has allowed us to develop sound and complete assume-guarantee rules for inferring the strongest safety properties satisfied by a collection of components (modelling e.g., networked systems and connectors) under the full collection of composition operators on the specification theory. Such compositional techniques are required in order to improve the efficiency with which the behaviour of a connector can be verified, which is of key importance when connectors are to be synthesised on the fly. Details are provided in Section 2.1.
- From (ii) we have developed a real-time extension of the specification theory [12] that allows us to model the real-time performance of networked systems. From this, we have formulated a technique based on quotient for synthesising connectors that are not only functionally compatible with their environment, but that are also responsive to the timing demands of the systems to be made interoperable. Details are provided in Section 2.2.
- Finally, strand (iii) has allowed us to test the suitability of our framework in meeting the objectives of CONNECT. We have developed an example based on the GMES scenario, for which we synthesise connectors in order to allow successful communication between a number of networked systems. This example has been applied to both the non-quantitative assume-guarantee framework for safety properties, as well as the real-time extension of the specification theory.

The second stream of work highlights intra- and inter-WP integration. We have clarified how the formalized *Connector Algebra* [11] relates to the specification theory (intra-WP2 integration), and we have demonstrated how this relationship correlates with the mediator patterns and the associated CONNECTor synthesis approach developed within WP3 [13] (inter-WP2-WP3 integration). In particular, we have connected the specification theory with the connector algebra so as to support WP3 synthesis. To this aim, we have provided a synthesis algorithm that produces a modular CONNECTor in terms of the primitives of the algebra and its composition operators. The modular CONNECTor is equivalent to its monolithic version as produced by the specification theory via quotient. An important aspect of our CONNECTor decomposition concerns the ability for it to be reflected in terms of constraints for the mapping-driven synthesis in WP3. Furthermore, the CONNECTed system, composed by the synthesized modular CONNECTor plus the considered NSs, satisfies a general goal representing crucial conditions for correct communication and coordination. These conditions can be considered as the *intent* of the synthesized CONNECTor or, equivalently, of the CONNECTed system. Details are provided in Section 2.3.

1.3 Review recommendations

The following list recalls the reviewers' recommendations for the work done within WP2 after the third year of the project. For each recommendation, the reactions of the WP2 participants are reported.

- *...There seem to be some inconsistencies between the synthesis approaches of WP2 and WP3. Deliverable D3.3 says that the connectors generated there are more specific and more concise than those of WP2. This needs to be explained better. Also ontology alignment in WP3 looks different from the one in WP2. This needs to be clarified...*

  We have integrated the specification theory with the connector algebra so as to support WP3 synthesis. In particular, we have provided an algorithm for the automated synthesis of modular CONNECTors, each of them expressed as the composition of independent mediators. A modular CONNECTor, as synthesized by our method, supports CONNECTor evolution and performs correct mediation.

To show correctness, we have formally defined the semantics of protocols (as well as of mediators and CONNECTors) by using a revised version of the usual Interface Automata (IA) theory. Then, we have proved that a modular CONNECTor for two protocols P and R enjoys the same correctness properties of the monolithic CONNECTor obtained by expressing the synthesis problem as a quotient problem between P and R, as specified by the WP2 specification theory. Concerning the set of considered mediation patterns and, hence, CONNECTor modularization, our synthesis method relies on a revised version of the WP2 CONNECTor algebra. We have revised the original algebra by adding an iterator operator and by giving its semantics in terms of our revised IA theory.

To show how modular CONNECTors support evolution, we have used a case study in the e-commerce domain to illustrate that relevant changes can be applied on a modular CONNECTor by simply acting on its constituent mediators, without entirely re-synthesizing its protocol.

- *At the last review, this work package presented the draft of a connector algebra and a compositional approach based on assume-guarantee reasoning for probabilistic liveness and reward properties. The reviewers recommended at that review that the relationship between the calculus developed in WP2 and its application and usefulness to the other work packages be clarified and asked specifically how synthesis is supported by the calculus. Some further comments about the ability to deal with mismatches were made.*

  At this review, the work package presented a compositional specification theory for connector behavior and composition which is based on some of the earlier ideas and directions but which essentially replaces the earlier draft connector algebra. Deliverable D2.3 includes a description of a quotient operator and its application to synthesis. In addition, the theory includes actions and states that have data parameters. These elements permit the theory to provide an underlying basis for the mediator synthesis in WP3, the extended LTS in WP3 and the register automata in WP4. The specification theory enables a general description of mismatches as constraints, which address comments on the limited set of mismatch situations that could be dealt with previously.

  At the previous review, WP2 reported considerable progress in the area of Quantitative Compositional Verification based on assume-guarantee reasoning, and the reviewers commended this but commented on the relatively weak proof rules. In this review period, this work stream reported improvements and extensions to the learning framework for assume-guarantee reasoning that go some way towards addressing the reviewers' comments. In addition, work on a new assume-guarantee framework is reported. The verification performance of this new framework is as yet uncompetitive in relation to previous work. Furthermore, it is unclear where the contract-based verification which has been developed in the second year comes into play in that context.

  ...

  In summary, work stream (i) has developed a compositional specification theory that can underpin synthesis work in the rest of the project and in doing so has clearly addressed recommendations made in the last review. Work stream (ii) has addressed the reviewers' comments with respect to proof rules. However, there is very little evidence that relevance to the rest of the project is being addressed.

  The next phase of WP2 has a goal to integrate work streams (i) and (ii) through a quantitative extension of the specification theory and we would recommend that this be treated as a priority together with the need to demonstrate application of the Quantitative Compositional Verification to a problem drawn from the CONNECT case studies.

  In an effort to bridge workstreams (i) and (ii), we have formulated an assume-guarantee reasoning framework for safety properties based on the compositional component specification theory developed in D2.3. Component specifications are expressed as pairs of prefix-closed sets, an assumption and a guarantee. We define the notion of satisfaction between components and AG specifications, as well as substitutive refinement that preserves implementation containment. The refinement preorder is linear-time, to also align with WP4. We also give compositional rules on AG specifications for the operations of parallel composition, conjunction and quotient, and apply the framework to a case study from the systems domain, where we demonstrate how components can be inferred at

run-time from their specifications and substituted without introducing errors. An extension of the AG framework with liveness properties (via quiescence) is envisaged.

Additionally, we have developed a quantitative, real-time extension of the component specification theory in order to integrate workstreams (i) and (ii) and to align with WP5 (latency). The framework is capable of reasoning about safety and bounded-liveness properties and admits as a refinement relation the largest pre-congurence preserving absence of safety and bounded-liveness errors. It supports the operators of parallel composition, conjunction, disjunction and quotient, which are compositional. The framework relies on the notion of time-lock, which corresponds to the stopping of the global system clock and results in strong algebraic properties. We apply the formalism to a case study from the systems domain, demonstrating how components can be synthesized from real-time specifications as in WP3, including also at runtime.

Continuing the work on a quantitative, real-time extension of the specification theory, we have formulated a realizable variant which is restricted to systems that are free of time-lock. The formalism includes the operators of parallel, conjunction, disjunction and quotient, and enjoys strong algebraic properties, as well as offering strong potential for implementability.

As the first step towards obtaining a quantitative, probabilistic and real-time, specification theory, which fully integrates workstreams (i) and (ii), we formulate the notion of refinement for a specification theory based on abstract probabilistic timed automata. These correspond to abstract specifications of components that exhibit both probability as well as real-time behaviour. Refinement in this setting is based on modal may/must specifications, rather than linear time. We develop the operators of parallel composition and conjunction and prove their compositionality properties. As future work, we intend to formulate disjunction and quotient.

1.4 Outline

This deliverable is organized as follows. In Chapter 2, we briefly discuss the work undertaken by WP2 in the fourth year of the project, and highlight the significance of the devised formalisms/theories with respect to their support for WPs 3-5. The work of Year 4 integrates, extends and finalizes previous WP2 work, meaning Chapter 2 should be interpreted as a chapter highlighting the contributions of the work package as a whole. In Chapter 3, we report an overall evaluation of the WP2 results with respect to the objectives and assessment criteria discussed in Deliverable D6.3 [14], hence refining the preliminary evaluation contained in that deliverable. The key conclusions of our work are stated in Chapter 4. Appendix A contains peer-reviewed papers related to the discussed work, hence providing more details with respect to the overview given in Chapter 2. Appendix B briefly reports the technical details for computing the $\perp$-backpropagation technique discussed in Section 2.2.

2 Year 4 work

In this chapter, we provide a brief overview of the two work streams undertaken by WP2 in the fourth year of the project. A detailed description of these works can be found in Appendix A, where the related CONNECT papers are reported. We recall that the discussed work integrates, extends and finalizes the WP2 work undertaken during the previous years of the project, meaning that this chapter highlights the contributions of WP2 overall. Outlining this chapter, Section 2.1 begins by introducing an assume-guarantee framework for reasoning about safety properties satisfied by components modelled using the specification theory, while Section 2.2 defines a real-time quantitative extension of the specification theory. In Section 2.3, we provide an overall discussion of how we connected the specification theory with the connector algebra so as to support WP3 synthesis, along with an informal description of the underlying algorithm used for the automated synthesis of modular CONNECTors.

2.1 Safe assume-guarantee reasoning framework for the compositional specification theory

In this section, we outline the development of a compositional assume-guarantee (AG) reasoning framework for the preservation of safety properties satisfied by components modelled using the specification theory reported in Years 2 and 3 of the project [9]. We begin by briefly recalling the essential concepts of the specification theory for modelling components, highlight the key features of the assume-guarantee reasoning framework, and conclude by evaluating our framework against the objectives of CONNECT by means of a small case-study/example.

In the context of CONNECT, the purpose of developing such an AG reasoning framework is strongly related to the necessity of ensuring that connected systems satisfy safety properties, e.g., a communication mismatch will never occur. For complex systems of considerable size, inferring such properties on the whole state-space is infeasible, due to issues of scalability. Instead, we need a compositional framework in which properties can be checked on the components of the system, and from which global properties can be deduced under a collection of sound and complete AG rules.

2.1.1 Primer on the specification theory

Recall from D2.3 [12] that a component model encodes the temporal ordering of interactions between the component and the environment. Interactions in the specification theory correspond to synchronisation of input and output actions, with the understanding that outputs are non-blocking. The input/output (I/O) type of an action is indicated by the interface of the component. We introduced a trace-based formalism for components, referred to as declarative specifications, and an operational representation referred to as Logic IOLTSs (I/O labelled transition systems). Component models are not required to be input-enabled (as in the I/O automata due to Lynch and Tuttle [26], and Jonsson [23]), making our formalism conceptually similar to the interface automata of De Alfaro and Henzinger [17, 18], which place assumptions on the behaviour of the environment.

To support a full specification theory, we introduced a refinement preorder corresponding to safe substitutivity. A declarative specification $\mathcal{Q}$ is a refinement of declarative specification $\mathcal{P}$, written $\mathcal{Q} \sqsubseteq_{dec} \mathcal{P}$, iff for any environment $\mathcal{E}$ composable with $\mathcal{P}$, if $\mathcal{P}$ and $\mathcal{E}$ can exist without introducing communication mismatches (i.e., $\mathcal{P}$ and $\mathcal{E}$ will never issue an output when the other is unwilling to receive it), then $\mathcal{Q}$ and $\mathcal{E}$ must not generate communication mismatches. Naturally, this definition of refinement is the weakest preorder preserving substitutivity. A similar notion of refinement can be defined on Logic IOLTSs (say P, Q), which we denote by $\sqsubseteq_{op}$. By giving a semantic preserving mapping $[\![\cdot]\!]^*$ from Logic IOLTSs to declarative specifications respecting $\mathrm{P} \sqsubseteq_{op} \mathrm{Q}$ iff $[\![\mathrm{P}]\!]^* \sqsubseteq_{dec} [\![\mathrm{Q}]\!]^*$, it is sufficient to consider only declarative specifications, which have more elegant algebraic properties. Henceforth, we will talk only of declarative specifications, and refer to such models as components.

Let $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{R}$ be components. The specification theory comes equipped with a range of compositional operators for constructing new components:

- **Parallel composition** $\mathcal{P} \mid\mid \mathcal{Q}$**.** This operator is used for examining the structural behaviour of $\mathcal{P}$ and $\mathcal{Q}$. Communication mismatches arising between $\mathcal{P}$ and $\mathcal{Q}$ are explicitly represented in $\mathcal{P} \mid\mid \mathcal{Q}$.

- **Conjunction** $\mathcal{P} \wedge \mathcal{Q}$**.** The conjunctive operation is the meet operator on the refinement preorder, meaning $\mathcal{P} \wedge \mathcal{Q} \sqsubseteq_{dec} \mathcal{P}$, $\mathcal{P} \wedge \mathcal{Q} \sqsubseteq_{dec} \mathcal{Q}$, and $\mathcal{R} \sqsubseteq_{dec} \mathcal{P}$ and $\mathcal{R} \sqsubseteq_{dec} \mathcal{Q}$ implies $\mathcal{R} \sqsubseteq_{dec} \mathcal{P} \wedge \mathcal{Q}$. As a result, $\mathcal{P} \wedge \mathcal{Q}$ is a component that does not introduce communication mismatches in any environment safe for $\mathcal{P}$ or $\mathcal{Q}$. Consequently, conjunction supports independent development by combining requirements.

- **Quotient** $\mathcal{R}/\mathcal{P}$**.** Given $\mathcal{R}$ and a sub-component $\mathcal{P}$, if there exists $\mathcal{Q}$ such that $\mathcal{P} \mid\mid \mathcal{Q} \sqsubseteq_{dec} \mathcal{R}$, then $\mathcal{P} \mid\mid (\mathcal{R}/\mathcal{P}) \sqsubseteq_{dec} \mathcal{R}$ and $\mathcal{Q} \sqsubseteq_{dec} \mathcal{R}/\mathcal{P}$. Thus, quotient can be thought of as the adjoint of parallel composition, and has applications to incremental development in the form of component synthesis.

- **Disjunction** as the dual of conjunction can also be added in a straightforward manner, as can hiding of actions, which supports abstraction and hierarchical development. In the case of the latter, the definition of the hiding operator depends on the I/O type of the action to be hidden.

For each of these operations, we prove a range of compositionality results, and highlight their algebraic properties. Besides the formal definition of a component, this level of detail is sufficient to understand the subsequent sections of this report, although a more detailed exposition is contained in [9]. Before concluding this section, we briefly state the definition of a component.

Definition 2.1 *A component $\mathcal{P}$ is a tuple $\langle \mathcal{A}^I_\mathcal{P}, \mathcal{A}^O_\mathcal{P}, T_\mathcal{P}, F_\mathcal{P} \rangle$, where $\mathcal{A}^I_\mathcal{P}$ and $\mathcal{A}^O_\mathcal{P}$ are the inputs and outputs respectively, which make up the interface of $\mathcal{P}$, and $T_\mathcal{P}$ and $F_\mathcal{P}$ are sets of traces over $(\mathcal{A}^I_\mathcal{P} \cup \mathcal{A}^O_\mathcal{P})$, referred to as the observable traces and inconsistent traces respectively, satisfying the following properties:*

- $T_\mathcal{P}$ *is prefix closed*
- $T_\mathcal{P}(\mathcal{A}^I_\mathcal{P})^* \subseteq T_\mathcal{P}$ *i.e., $T_\mathcal{P}$ is closed under input extensions*
- $F_\mathcal{P}(\mathcal{A}_\mathcal{P})^* \subseteq F_\mathcal{P}$ *i.e., $F_\mathcal{P}$ is closed under all extensions*
- $F_\mathcal{P} \subseteq T_\mathcal{P}$ *i.e., any inconsistent trace is also a permissible trace.*

$T_\mathcal{P}$ contains all observable interactions between the environment and $\mathcal{P}$. As inputs are issued by the environment, $\mathcal{P}$ cannot prevent them from being observed, meaning that $T_\mathcal{P}$ must be receptive on inputs. To cater for the unwillingness of $\mathcal{P}$ to accept a particular input from the environment, we treat traces with a non-enabled input as being inconsistent. Once an inconsistency has been encountered, the resultant behaviour is unspecified, so we allow for chaotic observations, which is why $F_\mathcal{P}$ is closed under arbitrary extensions.

2.1.2 Assume-guarantee reasoning framework

Based on the specification theory outlined in the previous section, we devised an assume-guarantee (AG) framework for reasoning about the safety properties satisfied by components. For this framework, we prove a number of sound and complete AG rules that allow one to infer the strongest safety property satisfied by the composition of any two components under the operations of the specification theory. The motivation for such a theory is heavily geared towards combatting issues of complexity and state space explosion during system development and verification.

The primitive objects of the framework are AG specifications (or contracts), which we write as a tuple $\langle \mathcal{A}^I, \mathcal{A}^O, \mathcal{R}, \mathcal{G} \rangle$, where $\mathcal{A}^I$ and $\mathcal{A}^O$ make up the interface of the specification (i.e., specifies the inputs and outputs that the specification deals with), and $\mathcal{R}$ and $\mathcal{G}$ are prefix closed sets of traces over $\mathcal{A}^I \cup \mathcal{A}^O$, referred to as the assumption and guarantee respectively. A specification represents a collection of components, precisely those components that satisfy the specification, which are said to be implementations. A component $\mathcal{P}$ is an implementation of the specification $\mathcal{S} = \langle \mathcal{A}^I_\mathcal{S}, \mathcal{A}^O_\mathcal{S}, \mathcal{R}_\mathcal{S}, \mathcal{G}_\mathcal{S} \rangle$, written $\mathcal{P} \models \mathcal{S}$, just if (leaving interfaces aside) any permissible trace of the component that is contained within the assumption is also contained within the guarantee, and does not allow the component to become inconsistent (i.e., $\mathcal{R} \cap T_\mathcal{P} \subseteq \mathcal{G} \cap \overline{F_\mathcal{P}}$, where $T_\mathcal{P}$ corresponds to all of $\mathcal{P}$'s traces, while $F_\mathcal{P}$ is all of $\mathcal{P}$'s inconsistent traces).

Given the definition of satisfaction, we say that specification $\mathcal{S}$ is a refinement of specification $\mathcal{T}$, written $\mathcal{S} \sqsubseteq \mathcal{T}$, just if the implementations of $\mathcal{S}$ are a subset of $\mathcal{T}$'s i.e., $\{\mathcal{P} : \mathcal{P} \models \mathcal{S}\} \subseteq \{\mathcal{P} : \mathcal{P} \models \mathcal{T}\}$. Equipped with refinement, we define the compositional operators of the specification theory directly on AG specifications. Below, we explain the intuition behind these operators.

Parallel composition. The parallel composition of two specifications is the strongest specification satisfying independent implementability. That is to say:

- For each $\mathcal{P}$, $\mathcal{Q}$: $\mathcal{P} \models \mathcal{S}$ and $\mathcal{Q} \models \mathcal{T}$ implies $\mathcal{P} \,||\, \mathcal{Q} \models \mathcal{S} \,||\, \mathcal{T}$.
- For each $\mathcal{P}$, $\mathcal{Q}$: if $\mathcal{P} \models \mathcal{S}$ and $\mathcal{Q} \models \mathcal{T}$ implies $\mathcal{P} \,||\, \mathcal{Q} \models X$, then $\mathcal{S} \,||\, \mathcal{T} \sqsubseteq X$.

The actual definition of the operator is included in CJK12, as reported in Appendix A, and is based on the well-known result due to Abadi and Lamport [1, 2]. At a high-level, the composition will only guarantee what can be guaranteed by both specifications, and the guarantee of one specification must not violate the assumption of the other specification. Based on these properties, we have the following sound and complete AG rule (leaving interfaces aside):

$$\textsc{Parallel}\ \frac{\mathcal{P} \models \mathcal{S} \quad \mathcal{Q} \models \mathcal{T} \quad \mathcal{S} \,||\, \mathcal{T} \sqsubseteq \mathcal{U}}{\mathcal{P} \,||\, \mathcal{Q} \models \mathcal{U}}.$$

Although the rule in [1] (which is conceptually similar to ours) is sound, it is not complete. We obtain both soundness and completeness in our framework by the assumption that the outputs of the specifications to be composed are disjoint, as an output should be controlled by at most one component. Adopting this convention allows us to break circularity in the AG rule, because a guarantee cannot be simultaneously violated by two components.

Conjunction. As for conjunction on components, we wish conjunction on specifications to correspond to the meet operator, but on $\sqsubseteq$. Consequently, the implementation set of $\mathcal{S} \wedge \mathcal{T}$ is equal to the intersection of the implementation sets of $\mathcal{S}$ and $\mathcal{T}$. Based on this, we can formulate a sound and complete assume-guarantee rule for conjunction (disregarding issues with interfaces).

$$\textsc{Conjunction}\ \frac{\mathcal{P} \models \mathcal{S} \quad \mathcal{Q} \models \mathcal{T} \quad \mathcal{S} \wedge \mathcal{T} \sqsubseteq \mathcal{U}}{\mathcal{P} \wedge \mathcal{Q} \models \mathcal{U}}.$$

The definition of conjunction has a straightforward assumption that corresponds to the union of the assumptions of the specifications to be composed. The guarantee contains the union of the guarantees, but constrained to just those behaviours that do not violate the assumption/guarantee on the other specification.

Quotient. To mirror the specification theory on components, we also introduce a quotient operator directly on specifications for decomposing the parallel composition operator. If there exists $\mathcal{T}$ such that $\mathcal{S} \,||\, \mathcal{T} \sqsubseteq \mathcal{U}$, then $\mathcal{U}/\mathcal{S}$ is defined, $\mathcal{S} \,||\, (\mathcal{U}/\mathcal{S}) \sqsubseteq \mathcal{U}$ and $\mathcal{T} \sqsubseteq \mathcal{U}/\mathcal{S}$. From this, we can formulate a sound and complete AG rule for quotient:

$$\textsc{Quotient}\ \frac{\forall \mathcal{P} \cdot \mathcal{P} \models \mathcal{S} \text{ implies } \mathcal{P} \,||\, \mathcal{Q} \models \mathcal{U}}{\mathcal{Q} \models \mathcal{U}/\mathcal{S}}.$$

Although we have not considered it is this report (the technical details being in CJK12, contained in Appendix A), this rule is only sound and complete when satisfaction of a specification by a component equates interfaces. This is related to the fact that parallel composition is only monotonic under refinement when certain restrictions are imposed on the interfaces of the components to be composed.

However, based on this definition of quotient, we can rewrite the AG rule for parallel composition to make use of the decomposition under quotient. This is particularly useful for system development, as we will often have the specification of a global system, rather than specifications of the systems to be composed.

$$\textsc{Parallel-Decompose}\ \frac{\mathcal{P} \models \mathcal{S} \quad \mathcal{Q} \models \mathcal{T} \quad \mathcal{T} \sqsubseteq \mathcal{U}/\mathcal{S}}{\mathcal{P} \parallel \mathcal{Q} \models \mathcal{U}}.$$

Compared to the rules for parallel composition and conjunction, note that none of the quotient rules deal with the quotient operator on components. This is not a problem, because when dealing with specifications and implementations, it does not entirely make sense to talk about the quotient of implementations. Instead, we should take the quotient of the specifications representing the implementations, and then find an implementation satisfying the resulting specification, as after all, if we have a component satisfying the global specification, then there is no need to decompose it.

In CJK12 (reported in Appendix A), we also provide definitions for constructing the most-general component that satisfies a specification, and document how to create the characteristic specification for a component. These transformations are important for moving between models of the AG framework and models of the compositional specification theory.

2.1.3 Evaluation

We illustrate our AG framework on a simple example concerned with a portion of the GMES (Global Monitoring for Environment and Security) scenario. For geographical areas prone to fire-risk, it is common to install battery-powered cameras that can transmit an image of the surrounding area to a central control centre, where an operator can take appropriate action. This allows fire crews to head directly to the affected area, which can greatly reduce the spread of fire and its associated destruction.

The system as a whole is composed of a fire detector, system controller and the physical camera itself. Intuitively, the sensor signals when a *fire* is detected, and can be *reset* once the fire has been acknowledged. The controller, on the other hand, waits for the indication of a *fire* from the sensor, after which it instructs the camera to *capture* an image of the environment, and awaits the resultant *photo* to be sent. At this stage, the controller will signal to the sensor that the fire has been acknowledged, and so the detector may be *reset*. The camera itself waits for a *capture* request, after which it will send the *photo*.

We iteratively derive a design for the connector (corresponding to the system controller) by successively applying AG rules and constructions. At this stage, we are crafting a specification for the detector and controller by hand. Later, we will see how the specification can be adapted on-the-fly in order to handle detectors that expect to be interacted with in different ways. We start by making use of two specifications for the combined effect of the detector and system controller:

1. Spec1: If the number of *capture* requests is equal to or one greater than the number of *photo*s taken, then the number of *fire* detections must be equal to or one greater than the number of *capture* requests sent to the camera.

2. Spec2: If the number of *capture* requests is equal to or one greater than the number of *photo*s taken, then a *photo* must be taken before the detector can be *reset*, and the detector cannot be consecutively *reset* without a *photo* having been taken in between.

Spec1 and Spec2 can be represented by the AG specifications $\langle \mathcal{R}_{\mathsf{Spec}}, \mathcal{G}_{\mathsf{Spec1}} \rangle$ and $\langle \mathcal{R}_{\mathsf{Spec}}, \mathcal{G}_{\mathsf{Spec2}} \rangle$ respectively, where the assumptions and guarantees are depicted in Figure 2.1. For simplicity, we represent sets of traces by means of finite automata, and annotate states with an F to indicate that a trace becomes inconsistent. The combined effect of Spec1 and Spec2 is given by the conjunctive specification $\mathsf{Spec1} \wedge \mathsf{Spec2} = \langle \mathcal{R}_{\mathsf{Spec}}, \mathcal{G}_{\mathsf{Spec1} \wedge \mathsf{Spec2}} \rangle$, the guarantee of which is presented in Figure 2.2.

We now demonstrate compositional AG reasoning on our specifications (references to definitions, lemmas and theorems in the following paragraph refer to CJK12, reproduced in Appendix A). By Definition 10 we can find implementations $\mathcal{I}(\mathsf{Spec1})$ and $\mathcal{I}(\mathsf{Spec2})$ of Spec1 and Spec2 respectively, which by Theorem 5 allows us to derive $\mathcal{I}(\mathsf{Spec1}) \wedge \mathcal{I}(\mathsf{Spec2}) \models \mathsf{Spec1} \wedge \mathsf{Spec2}$. According to Lemma 4, this means that $\mathcal{I}(\mathsf{Spec1}) \wedge \mathcal{I}(\mathsf{Spec2}) \sqsubseteq_{imp} \mathcal{I}(\mathsf{Spec1} \wedge \mathsf{Spec2})$. Now by Theorem 3, we know $\mathsf{Spec1} \wedge \mathsf{Spec2} \sqsubseteq \mathsf{Spec1}$, so from Lemma 2 we obtain $\mathcal{I}(\mathsf{Spec1} \wedge \mathsf{Spec2}) \models \mathsf{Spec1}$, and from Lemma 4 we derive $\mathcal{I}(\mathsf{Spec1} \wedge \mathsf{Spec2}) \sqsubseteq_{imp} \mathcal{I}(\mathsf{Spec1})$. By similar reasoning it can be shown that $\mathcal{I}(\mathsf{Spec1} \wedge \mathsf{Spec2}) \sqsubseteq_{imp} \mathcal{I}(\mathsf{Spec2})$, hence by Theorem 2 of [9] we acquire $\mathcal{I}(\mathsf{Spec1} \wedge \mathsf{Spec2}) \sqsubseteq_{imp} \mathcal{I}(\mathsf{Spec1}) \wedge \mathcal{I}(\mathsf{Spec2})$. Mutual refinement of components in our framework corresponds to equality of models, so $\mathcal{I}(\mathsf{Spec1} \wedge \mathsf{Spec2}) = \mathcal{I}(\mathsf{Spec1}) \wedge \mathcal{I}(\mathsf{Spec2})$. Such an

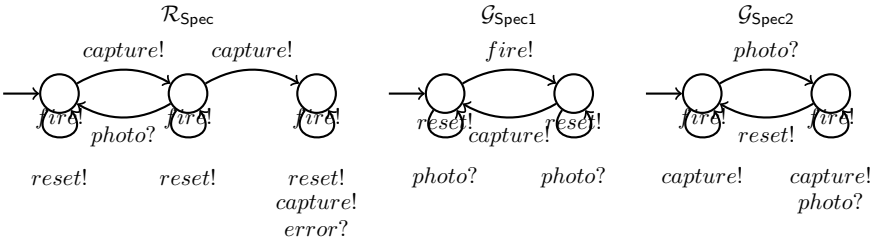


Figure 2.1: Assumption and guarantees for Spec1 **and** Spec2

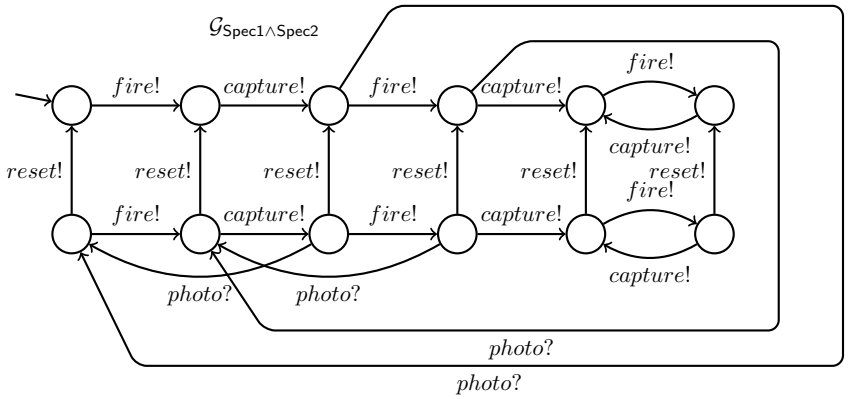


Figure 2.2: The guarantee for Spec1 $\wedge$ Spec2

implementation is shown in Figure 2.3. Note how this component is unwilling to *capture* an image after encountering two *fire* signals without a *photo* being taken. This is because $\mathcal{R}_{\mathsf{Spec}}$ can issue an *error* after such an occurrence, but this is not accepted by $\mathcal{G}_{\mathsf{Spec1}\wedge\mathsf{Spec2}}$. Moreover, this implementation is able to see an unbounded number of *fire* indications that it *capture*s an image for, without ever having to *reset* the detector.

We now demonstrate how our theory can be used in the context of CONNECT in order to support on-the-fly synthesis of connectors. By using quotient on the connector formulated in the previous part by means of conjunction, we now allow the environment to change by insisting that the detector must be *reset* before it can signal any new *fire*s. We do so by making use of a constraint specification $\mathsf{Detect} = \langle \mathcal{R}_{\mathsf{Detect}}, \mathcal{G}_{\mathsf{Detect}} \rangle$ that requires *fire* and *reset* to alternate (shown in Figure 2.4). We wish to find an implementation for the system controller, let it be called Controller, such that Controller is an implementation of Spec1 $\wedge$ Spec2 subject to the constraints imposed by Detect. This is equivalent to requiring $\mathsf{Controller} \models (\mathsf{Spec1} \wedge \mathsf{Spec2})/\mathsf{Detect}$. The specification $(\mathsf{Spec1} \wedge \mathsf{Spec2})/\mathsf{Detect}$ is exhibited in Figure 2.5, and the most general implementation is obtained from $\mathcal{G}_{(\mathsf{Spec1}\wedge\mathsf{Spec2})/\mathsf{Detect}}$ by appending all non-enabled inputs as inconsistent traces. In contrast to $\mathcal{I}(\mathsf{Spec1} \wedge \mathsf{Spec2})$, the constraints imposed by Detect on Spec1 $\wedge$ Spec2 means that any candidate implementation for Controller will ensure that there can be at most one outstanding *fire* signal that has not been *reset*.

2.2 Quantitative extension of the compositional specification theory and its application

A key requirement of CONNECT is to model the QoS aspects of networked systems, in addition to the functional behaviour. In this section, we focus on a quantitative extension of the compositional specification theory, which is capable of modelling the real-time performance of components and connectors. We define a substitutive refinement preorder that takes into account the timing constraints imposed by networked systems, and the full collection of compositional operators (as mentioned in Section 2.1.1), in order to see how the timing constraints of the individual components influence the interactive behaviour

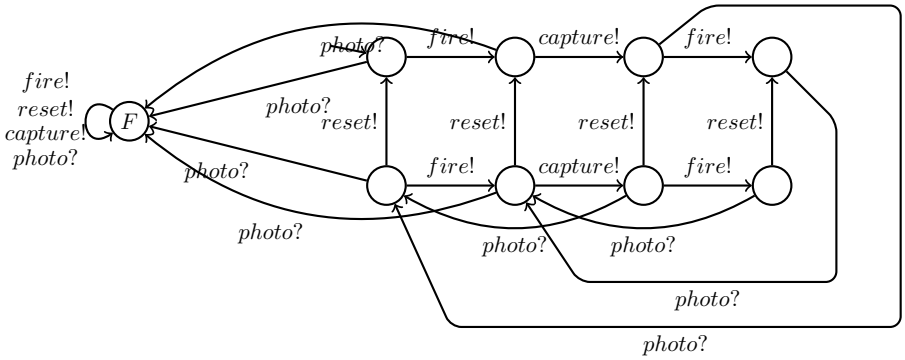


Figure 2.3: The most general implementation of Spec1 ∧ Spec2

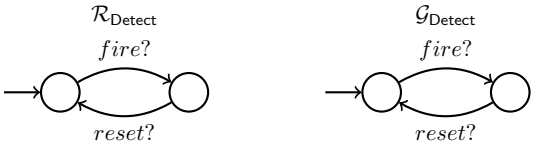


Figure 2.4: Specification of a restrictive detector constraint Detect

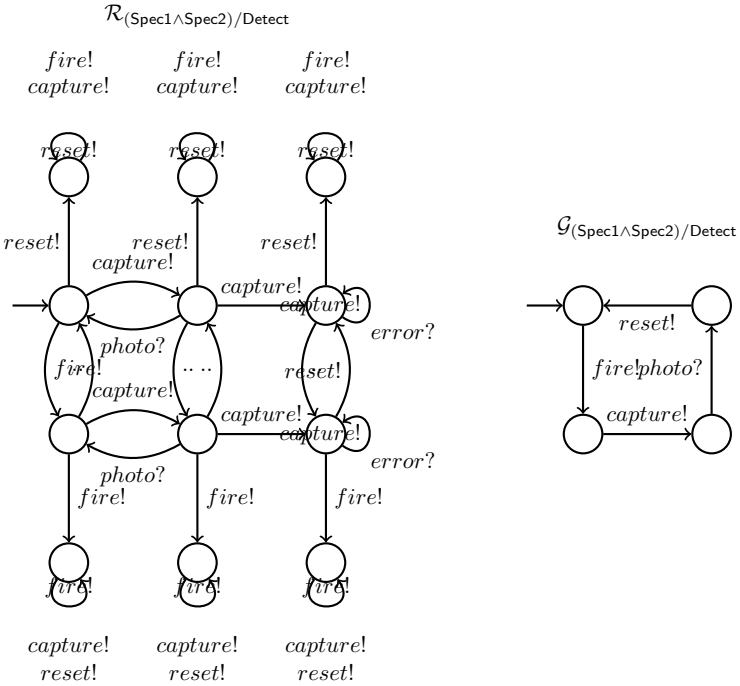


Figure 2.5: Specification for (Spec1 ∧ Spec2)/Detect

of the system as a whole.

2.2.1 Component model

At the heart of our theory are timed I/O automata, an extension of the timed automata of Alur and Dill [3], in which actions are partitioned into inputs and outputs, and states are annotated with *co-invariants* in addition to invariants. While invariants specify the bounds beyond which time may not progress, co-invariants specify the bounds beyond which the component will time-out and enter an inconsistent state.

Thus, invariants specify liveness timing guarantees on the outputs in a particular state, while co-invariants express liveness timing assumptions on the inputs enabled in a state. Similarly, the guard on an input transition specifies a safety timing assumption, while the guard on an output labelled transition specifies a safety timing guarantee. Thus, our models allow to capture the AG nature of timed components.

Informally, the parallel composition of two components automatically checks whether the guarantees provided by one component meet the assumptions required by the other. For instance, the unexpected arrival of an input at a particular location and time (indicated by a non-enabled transition) leads to a *safety error* in the parallel composition. On the other hand, non-arrival of an expected input at a location before its time-out (specified by the co-invariant) leads to a *bounded-liveness error* in the parallel composition.

Our formalism differs from the framework due to De Alfaro *et al.* [19], in that we take as our refinement relation a weakest preorder corresponding to substitutivity, rather than timed alternating simulation. Moreover, by equating safety and bounded liveness errors with the single notion of inconsistency, we are able to avoid the necessity of utilising two transition systems. Our framework generalises that of [19] by supplying compositional definitions of conjunction, disjunction and quotient, which are crucial for combining requirements of networked systems in order to synthesise connectors on-the-fly.

The timed I/O framework due to Larsen *et al.* [15] also makes use of timed alternating refinement, which has lower complexity than our relation, but does not admit the weakest pre-congruence for safe substitutivity. The framework does, however, include definitions for conjunction and quotient. A shortcoming of the framework is that errors must be specified by the user, which means that the compositional operators and refinement are error-agnostic.

2.2.2 Semantic model

The semantics of our component models are given in terms of timed I/O transition systems (TIOTSs), which provide an explicit representation for the timed transitions of the automaton. A configuration of a TIOTS is a location and clock-valuation pair, or one of the special chaotic states $\bot$ and $\top$. $\bot$ is used to represent inconsistency (i.e., violation of safety and bounded-liveness properties), while $\top$ represents timestop (i.e., a point from which time may not progress).

The TIOTS for a timed component is largely generated in the same way as for timed automata, except that we must give special consideration to the invariant and co-invariant. Let $\longrightarrow_S$ be the standard transition system for a timed component (i.e., treating the component as a timed automaton), then the TIOTS representation $\longrightarrow$ is the smallest relation such that:

- If $(l,t) \xrightarrow{a}_S (l',t')$ and:
  - t' satisfies the invariant and co-invariant on l', then $(l,t) \xrightarrow{a} (l',t')$
  - t' satisfies the invariant, but does not satisfy the co-invariant on l', then $(l,t) \xrightarrow{a} \bot$
  - t' does not satisfy the invariant on l', then $(l,t) \xrightarrow{a} \top$
- If $(l,t) \xrightarrow{d}_S (l,t+d)$ and:
  - $t+d$ satisfies the invariant and co-invariant on l, then $(l,t) \xrightarrow{d} (l,t+d)$
  - there exists $0 < \delta \leq d$ such that δ satisfies the invariant, but not the co-invariant on l, then $(l,t) \xrightarrow{d} \bot$
- If $(l,t) \not\xrightarrow{a}_S$ with a an input, then $(l,t) \xrightarrow{a} \bot$
- If $(l,t) \not\xrightarrow{a}_S$ with α an output or delay, then $(l,t) \xrightarrow{\alpha} \top$.

The initial state of the TIOTS is $(l_0,0)$ if both the invariant and co-invariant hold in this configuration, is $\top$ if the invariant does not hold on $(l_0,0)$, and is $\bot$ otherwise. Note that every state of the TIOTS has outgoing transitions for each input, output and delay, and also that the TIOTS is time-additive, meaning that it satisfies the triangle rule on delays.

Table 2.1: State representations under composition operators

$\|\|$	$\top$	p	$\bot$
$\top$	$\top$	$\top$	$\top$
q	$\top$	$p{\times}q$	$\bot$
$\bot$	$\top$	$\bot$	$\bot$

$\wedge$	$\top$	p	$\bot$
$\top$	$\top$	$\top$	$\top$
q	$\top$	$p{\times}q$	q
$\bot$	$\top$	p	$\bot$

$\vee$	$\top$	p	$\bot$
$\top$	$\top$	p	$\bot$
q	q	$p{\times}q$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$

$\%$	$\top$	p	$\bot$
$\top$	$\bot$	$\bot$	$\bot$
q	$\top$	$p{\times}q$	$\bot$
$\bot$	$\top$	$\top$	$\bot$

$\neg$	
$\top$	$\bot$
p	p
$\bot$	$\top$

In many practical applications, including in CONNECT, the presence of timestop ($\top$) is an undesirable feature, as we should never halt the progress of time. Any TIOTS for which there is no strategy of the component allowing it to avoid $\top$ is said to be unrealisable. For such TIOTSs, it is inevitable that $\top$ will be reached for some strategy of the environment, and so time will be forced to stop. We develop two frameworks for timed systems, one where the stopping of time is permitted, and another where time may not be stopped. The latter builds on the former by using $\bot$ and $\top$ back propagation techniques. In the sequel, we consider the first framework, until we state otherwise.

2.2.3 Refinement

Substitutive refinement of timed components is characterised by a test that checks for preservation of $\bot$-presence under every environment. Formally, $\mathcal{Q}$ is a refinement of $\mathcal{P}$, written $\mathcal{Q} \sqsubseteq \mathcal{P}$, iff for each environment $\mathcal{R}$ of $\mathcal{P}$[1], if $\bot$ is reachable in $\mathcal{Q} \parallel \mathcal{R}$, then $\bot$ is reachable in $\mathcal{P} \parallel \mathcal{R}$ (where $\parallel$ is defined in Section 2.2.4). It is obvious that such a relation is the weakest preorder corresponding to safe-substitutivity of components.

In CKW12 (presented in Appendix A), we show that in a game-based interpretation of the specification theory, refinement corresponds to strategy containment. Similarly, in a trace-based representation of timed components, refinement is characterised by means of trace containment. Both of these frameworks avoid the need to quantify over all environments. These different representations for timed components do not directly concern the modelling of networked systems in CONNECT, so we do not describe them in this summary. Instead, we focus on a simple operational model based on TIOTSs.

2.2.4 Compositional operations

The compositional operators of the specification theory (consisting of parallell $||$, conjunction $\wedge$, disjunction $\vee$ and quotient $\%$) are defined directly on TIOTSs. Although we have not considered it in CKW12 (see Appendix A), the operations can be defined on the automata representation by using symbolic techniques for timing constraints.

Given appropriate alphabetisations of TIOTSs $\mathcal{P}$ and $\mathcal{Q}$ (as for the non-quantitative specification theory), the composition under $\otimes \in \{||, \wedge, \vee, \%\}$, written $\mathcal{P} \otimes \mathcal{Q}$, can be defined in a straightforward manner, as prescribed by the following definition. Note that for the quotient operator, we require that $\mathcal{Q}$ is deterministic, which can be achieved by a modified subset construction.

Definition 2.2 *For suitable alphabetisations of TIOTSs $\mathcal{P}$ and $\mathcal{Q}$, $\mathcal{P} \otimes \mathcal{Q}$ is the TIOTS with set of states $(S_\mathcal{P} \times S_\mathcal{Q}) \cup S_\mathcal{P} \cup S_\mathcal{Q}$[2], initial state $s^0_\mathcal{P} \otimes s^0_\mathcal{Q}$, and transition relation $\longrightarrow$, which is the smallest relation containing $\longrightarrow_\mathcal{P}$ and $\longrightarrow_\mathcal{Q}$, and satisfying the following rules (when $p \otimes q \notin \{\top, \bot\}$):*

$$\frac{p \xrightarrow{\alpha}_\mathcal{P} p' \quad q \xrightarrow{\alpha}_\mathcal{Q} q'}{p \otimes q \xrightarrow{\alpha} p' \otimes q'} \qquad \frac{p \xrightarrow{a}_\mathcal{P} p' \quad a \notin A_\mathcal{Q}}{p \otimes q \xrightarrow{a} p' \otimes q} \qquad \frac{q \xrightarrow{a}_\mathcal{Q} q' \quad a \notin A_\mathcal{P}}{p \otimes q \xrightarrow{a} p \otimes q'},$$

where $p \otimes q$ has the interpretation given in Table 2.1, and $(l_p, t_p) \times (l_q, t_q) = ((l_p, l_q), t_p \uplus t_q)$ for location-valuation pairs. Note that the clocks of $\mathcal{P}$ and $\mathcal{Q}$ must be disjoint, so $t_p \uplus t_q$ corresponds to the union of valuation functions having disjoint domains.

[1] An environment for $\mathcal{P}$ is any timed component having a complementary alphabet to $\mathcal{P}$.

[2] It is assumed that $S_\mathcal{P} \cap S_\mathcal{Q} = \{\top, \bot\}$, where $S_\mathcal{P}$ and $S_\mathcal{Q}$ also contain the location-valuation pairs for $\mathcal{P}$ and $\mathcal{Q}$.

The definitions of the compositional operations are simplified by our representation of TIOTSs, in that every configuration-state has successors for each input, output and timed action (perhaps going to $\top$ or $\bot$), in addition to the encoded behaviour in Table 2.1.

As is well known in the literature, quotient can be defined as a derived operator in terms of parallel composition and mirroring [31, 21, 7], i.e., $\mathcal{P}\%\mathcal{Q} = (\mathcal{P}^{\neg} \parallel \mathcal{Q})^{\neg}$. The unary mirroring operator $^{\neg}$, must first determinise its argument, then switch the $\top$ and $\bot$ states, as reported in Table 2.1. This is equivalent to interchanging the invariant and co-invariant in the automaton.

In CKW12, we show that the operators of the specification theory satisfy the desired properties, in that $\wedge$ and $\vee$ are respectively the meet and join operations on the refinement preorder, and quotient is the least refined component whose parallel composition with $\mathcal{Q}$ is a refinement of $\mathcal{P}$.

2.2.5 Realisability

Under substitutive refinement, we have that $\bot$ is refined by any location-valuation configuration, which is in turn refined by $\top$. As previously remarked, $\top$ signifies the stopping of time, which is undesirable for CONNECT, as components must work continually in real-time. In this section, we remark on how our theory can be extended to handle only realisable components that will not encounter $\top$.

For any TIOTS $\mathcal{P}$, we can extract the most general TIOTS that is realisable, denoted $\mathcal{P}^R$, by means of $\top$-backpropagation. This is achieved by repeatedly equating location-valuation configurations with $\top$ on the determinised TIOTS $\mathcal{P}^D$ according to the following rules: $p = \top$ if

- **auto-$\top$.** $p \xrightarrow{a} \top$ when a is an input
- **semi-$\top$.** $p \xrightarrow{d} \top$, and $d' \leq d$ and $p \xrightarrow{d'} \xrightarrow{a}$ p' when a is an output implies $p' = \top$.

Note that $\top$-backpropagation treats as $\top$ any state from which the component cannot avoid reaching $\top$ when the environment acts as an adversary. Auto-$\top$ states must be marked as $\top$, because of the possibility that the environment will issue an input leading to $\top$. Semi-$\top$ states on the other hand, are states from which time stop will eventually occur if an output cannot be made by the component to a non-$\top$ state before the time stop occurs.

$\mathcal{P}$ is said to be realisable just if $\mathcal{P}^R$ is not the $\top$-TIOTS, meaning that the component is always able to avoid reaching $\top$. Refinement of realisable components is defined as previously, except that only realisable environments are considered. Parallel composition and disjunction, which maintain realisability, also remain as before. For conjunction, quotient and mirroring, the original definitions are also used, but it is necessary to first *normalise* the components to be composed. Normalisation can be thought of as performing $\bot$-backpropagation, which equates a location-valuation state p with $\bot$ just if:

- **auto-$\bot$.** $p \xrightarrow{a} \bot$ when a is an output
- **semi-$\bot$.** $p \xrightarrow{d} \bot$, and $d' \leq d$ and $p \xrightarrow{d'} \xrightarrow{a}$ p' when a is an input implies $p' = \bot$.

This means that a state is equivalent to $\bot$, if the component can issue a sequence of outputs leading to $\bot$, or if after a delay $\bot$ is encountered without the possibility of the environment issuing an input to avoid reaching this state. For a realisable component $\mathcal{P}$, we denote the normalised version by $\mathcal{P}^N$.

Equipped with normalisation, we now define conjunction as $(\mathcal{P}^N \wedge \mathcal{Q}^N)^R$, quotient as $(\mathcal{P}^N\%\mathcal{Q}^N)^R$ and mirroring as $(\mathcal{P}^N)^{\neg}$, providing the resulting component is not the $\top$-TIOTS. Otherwise, the operations are undefined. In the following section, we demonstrate how these compositional operators work in practice.

2.2.6 Evaluation

We use the GMES (Global Monitoring for Environment and Security) and Terrorist Alert scenario as the basis to evaluate and demonstrate the use of *timed quotient* in the synthesis of connectors *with quantitative constraints*. In the GMES scenario, one of the most crucial network services is the *video streaming service* between *video streaming sources* and *video streaming consumers*. The video streaming sources can be UAV/UGV cameras, fixed or controlled (e.g. PTZ) cameras for traffic monitoring or estate/environment surveillance, or mobile cameras for patrol. The video streaming consumers can be either staff in the Command and Control Fire Operations Center or fire fighters at the fire scenes. The video to be streamed

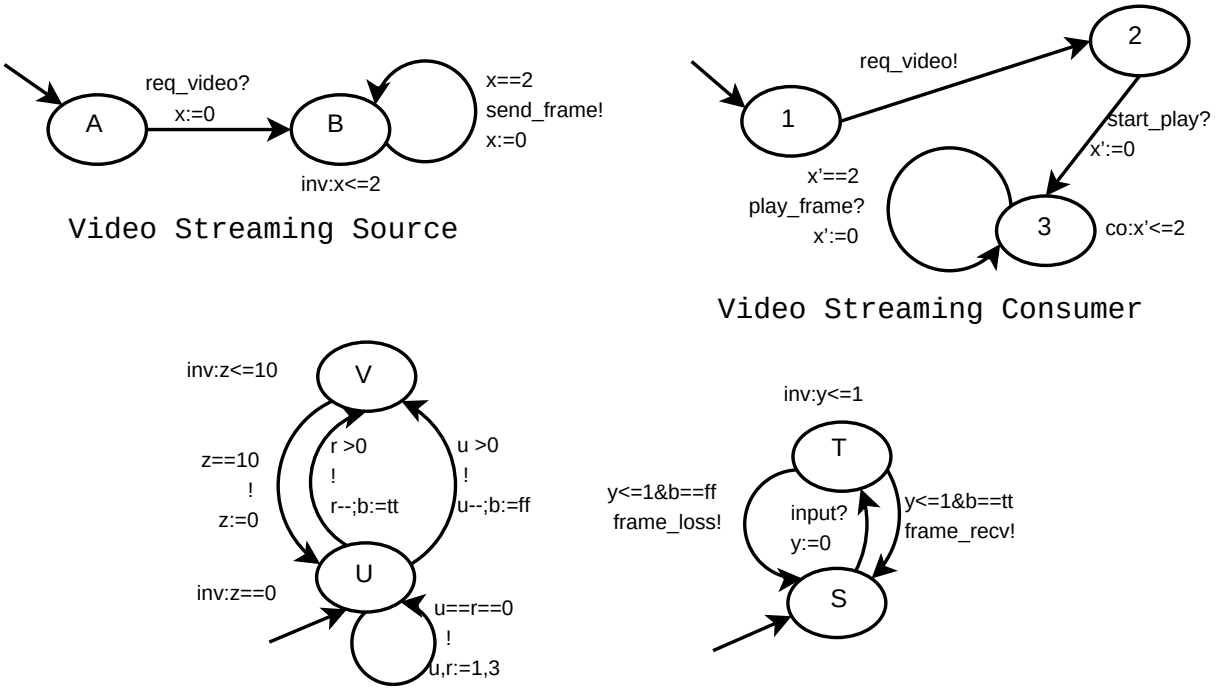


Figure 2.6: Video streaming services

can be either live or recorded (UAV/UGV or mobile patrol cameras), and encoded in various formats (e.g. MPEG/H.264).

Due to the unpredictability and volatility of a fire-fighting situation and its organisation (a typical example being on-the-scene reinforcement integration from another country), the GMES scenario calls for dynamic discovery and integration of maximal video resources available, which in turn requires on-the-fly synthesis and deployment of connectors to overcome the large heterogeneity in the dynamic context with lots of improvisations. One of the key responsibilities for video streaming connectors is the conversion and adaptation between different video formats and protocols, covering what we call the qualitative aspect. In the case study below, we demonstrate that our timed specification theory can synthesise the quantitative aspects (of the connectors) as well, i.e. the timing constraints employed in the connectors.

Below we abstract away some irrelevant details of the video streaming connectors and focus on the timing issues. The video streaming source is modelled as a timed automaton (`Video Streaming Source`, Figure 2.6) that, after being requested to stream video, sends out one frame every two time units. The video streaming consumer is modelled by a timed automaton (`Video Streaming Consumer`, Figure 2.6) that first sends the video streaming request to the source and then waits for a signal to start playing the video. Once the playback has started, the consumer expects that one frame will arrive every two time units (using co-invariants).

However, due to changeability in fire-fighting environments, the communication media between video streaming sources and consumers can vary greatly in QoS and reliability parameters. For instance, the communication media can be Wifi, cellular, satellite or wired networks or any hybrid combination of them. The QoS and reliability parameters of such media can have large variations, which can impact greatly on the quality and reliability of the streaming service. In order to bridge the variations, and guarantee the quality of streaming service no matter what communication media is used, the streaming connectors need to dynamically synthesise their timing constraints/parameters based on the actual QoS parameters collected from the communication media in use.

In this case study, we use a specific model with a fixed set of parameters for the communication media. However, both the model and parameters can be dynamically changed since the synthesis of

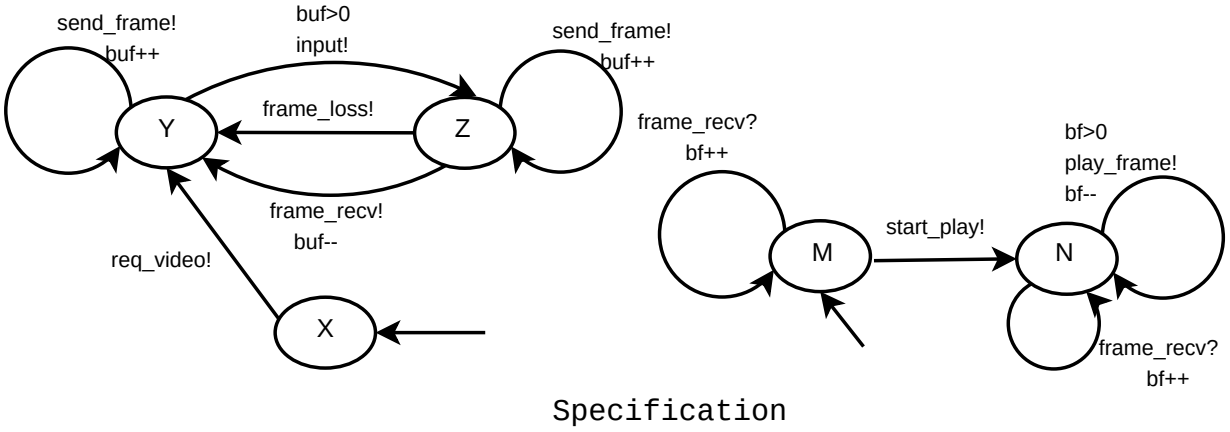


Figure 2.7: Specification of the services

timing-constraints can be done on-the-fly using arbitrary models and parameter sets.

The communication media is modelled using two timed automata, the combined effect of which we refer to as `Communication Media` (Figure 2.6). The factoring of `Communication Media` into two automata is solely for the purpose of simplifying the presentation. The left automaton models the fact that our communication media transmits through different phases of reliability in a periodic manner. We fix a period to have four time slots ($u + r = 4$) and each time slot to have ten time units ($z = 10$). For some time slots, the communication media is in the reliable mode V, while for others it is in the unreliable mode U. The users need only supply the ratio of reliable vs unreliable (i.e. u/r), which is $1/3$ for our case. Given a specific slot, its reliability or not (modelled as variable b) is chosen non-deterministically by the automaton. The right automaton, on the other hand, models the fact that, when a frame has been placed on the communication media, the outcome will be given at a time-point non-deterministically selected within one time unit. The outcome can be either *frame_loss* or *frame_recv* depending on the mode of the communication media. The model of the communication media is the product of the two automata.

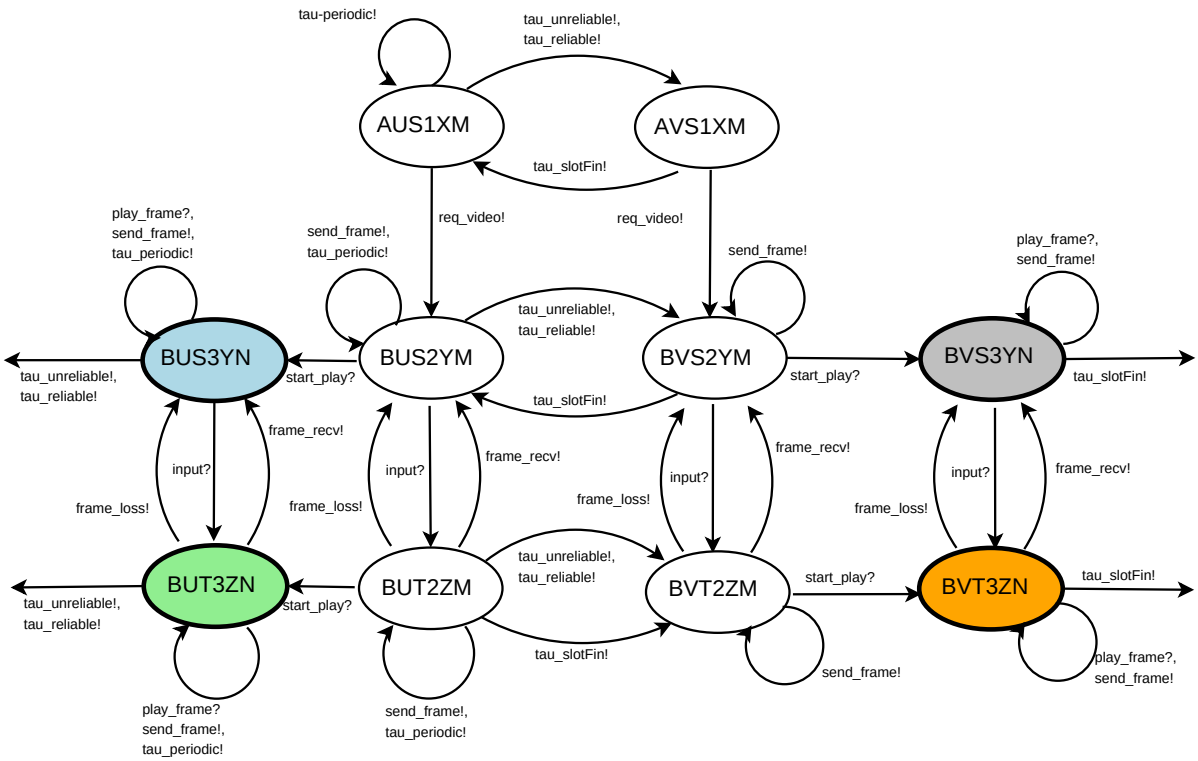
The overall specification of the streaming service is given by `Specification` (Figure 2.7), which is also factored into two automata, and captures the facts that: 1) the occurrence of actions needs to be ordered, e.g. *send_frame* after *req_video* and *frame_loss* / *frame_recv* after *input*; and 2) the number of frames received shall not exceed the number of frames sent. Note that there will need to be two buffers in the system, one on the source side buf and the other on the consumer side bf.

If the communication media is put in parallel directly with the video streaming source and consumer, then the system will not work very well as a whole. For instance, if the current frame buffering on the consumer side is empty and the communication media is in unreliable mode, then there will be no frame for the next playback. A possible strategy to work around this problem is to let the connector inform the consumer to *start_play* only after a proper delay, which is calculated based on the QoS parameters and current mode of the communication media, so that the consumer side can buffer a number of frames large enough to tide the consumer over during future unreliability of the communication media. The expected future behaviour of the communication media is captured in the values like u and r.

In total, our case study consists of six timed automata utilising four clock variables, four integer variables and one boolean variable. We now give the detailed steps to calculate the quotient:

$$\texttt{Specification} \,\%\, (\texttt{Video Streaming Source} \parallel \texttt{Video Streaming Consumer} \parallel \texttt{Communication Media}).$$

Our calculation is based on the equation $\mathcal{P}\%\mathcal{Q} = (\mathcal{P}^{\neg} \parallel \mathcal{Q})^{\neg}$, so we first construct the product $\texttt{Specification}^{\neg} \parallel (\texttt{Video Streaming Source} \parallel \texttt{Video Streaming Consumer} \parallel \texttt{Communication Media})$ (as depicted in Figure 2.8). To obtain the timed quotient automaton, further mirroring is performed on the product by first computing the $\perp$-backpropagation, after which the inputs and outputs, as well as invariants and co-invariants, must be exchanged. Note that $\top$-backpropagation is not needed here as the product automaton constructed is automatically realisable. The technical details for computing the $\perp$-backpropagation, along with the actual derivation, are included in Appendix B.



Initially, we have buf,bf:=0 b:=tt u,r:=0, x,y,z,x':=0,

tau_periodic!:	tau_reliable!:	tau_unreliable!:	tau_slotFin!:
u==r==0 u,r:=1,3	r >0 r--;b:=tt	u >0 u--;b:=ff	z==10 z:=0

req_video!:	send_frame!:	input?:	frame_recv!:	frame_loss!:	start_play?:	play_frame?:
x:=0	x==2 x:=0;buf++	buf>0 y:=0	y<=1&b=tt bf++,buf--	y<=1&b==ff	x':=0	x'==2&bf>0 bf--, x':=0

AUS1XM	inv:z==0	BVS2YM	inv:x<=2&z<=10	BVT2ZM	inv:x<=2&z<=10&y<=1		
AVS1XM	inv:z<=10	BVS3YN	co:x'<=2	inv:x<=2&z<=10	BVT3ZN	co:x'<=2	inv:x<=2&z<=10&y<=1
BUS3YN	co:x'<=2	inv:x<=2&z==0	BUT3ZN	co:x'<=2	inv:x<=2&z==0&y<=1		
BUS2YM	inv:x<=2&z==0	BUT2ZM	inv:x<=2&z==0&y<=1				

Global Invariants: 0<=x,x'<=2 0<=z<=10

Figure 2.8: The product automaton with its transitions/invariants/co-invariants specified below

Although the procedure for performing $\perp$-backpropagation has been applied manually in this example, it should be straightforward to see that it can be automated without much effort. The timing constraints synthesised are obviously subtle and highly non-trivial, which demonstrates the effectiveness of our quotienting technique for synthesising a connector in the GMES video streaming scenario, and its general applicability to networked systems with complex timing constraints.

2.3 Connecting the specification theory with the CONNECTor algebra to support WP3 synthesis

In this section, we give an informal overview of a method for the automated synthesis of modular CONNECTors. A detailed description of the method can be found in IT12 as reported in Appendix A. A modular CONNECTor is represented as a suitable composition of independent *mediators*. Each mediator can be seen as a basic (sub-)CONNECTor that realizes a specific mediation pattern, which corresponds to the solution of a recurring protocol mismatch. As mentioned in Section 1.3, our method for the automated synthesis of modular CONNECTors has been conceived to address two fundamental questions raised in the last reviews' recommendations:

Q1 - "how does the CONNECT*or algebra* relate to the *compositional specification theory* devised within WP2?";

Q2 - "how do the above WP2 work streams integrate with WP3 work on *automated* CONNECT*or synthesis*?".

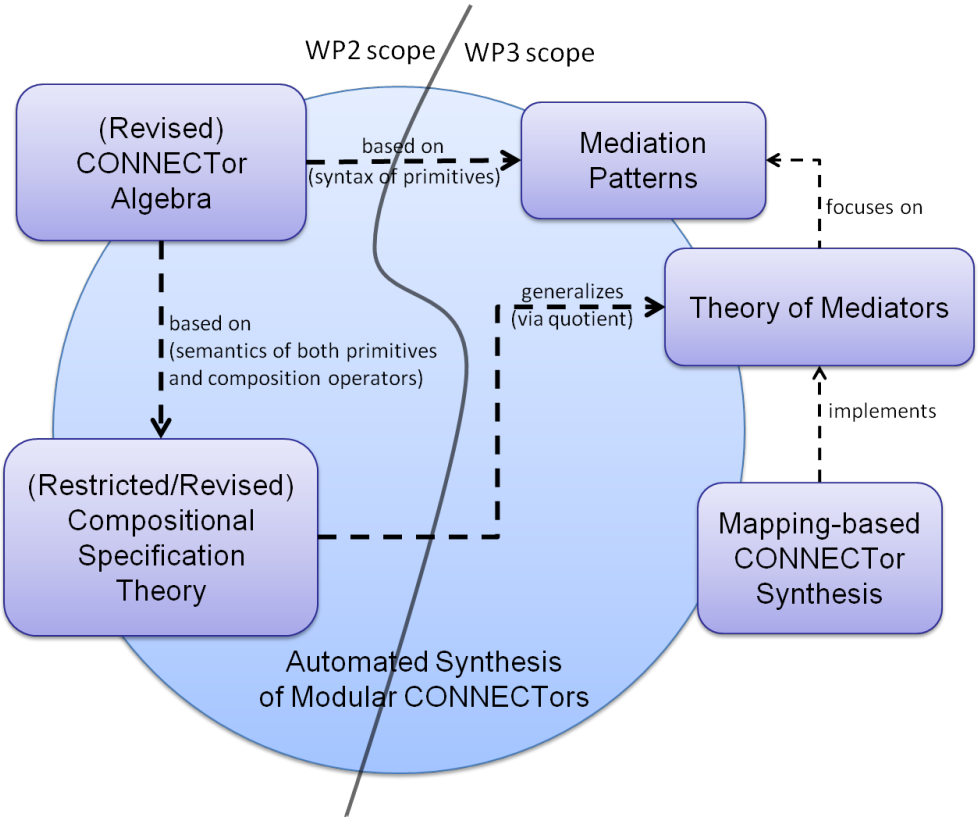


Figure 2.9: Overview of the intra-WP2 and inter-WP2-WP3 integration

As pictorially sketched in Figure 2.9, our method for the automated synthesis of modular CONNECTors relies on some fundamental pieces of work developed within both WP2 and WP3, and establishes some

relationships among them (see the thicker dashed arrows within the circle). In particular, each mediator synthesized as a basic constituent of a modular CONNECTor is an instance of a specific mediation pattern (e.g., reordering of messages) from the WP3 theory of mediators. To enable automated reasoning on our CONNECTor decomposition (e.g., to prove that a modular CONNECTor is equivalent to a WP3 monolithic CONNECTor, or is free of mismatches), we have formally defined the semantics of protocols, as well as of mediators and CONNECTors, by using a revised and restricted version of the WP2 compositional specification theory. It is restricted in the sense that we only consider deterministic protocols. Under this assumption, the WP2 compositional specification theory largely coincides with the *Interface Automata* (IA) theory [16]. Further revisions account for the notions of *hidden* actions and *semantically related* actions (yet syntactically different), which are needed for our synthesis algorithm, the details of which are contained in IT12 (see Appendix A).

Based on this revised version of the compositional specification theory, we prove that a modular CONNECTor for two protocols P and R enjoys the same correctness properties of the monolithic CONNECTor obtained by expressing the synthesis problem as the *quotient* problem between P and R [12]. Since, as described in D2.3 [12], CONNECTors synthesized through quotient generalizes WP3 CONNECTors, this also means that our modular CONNECTors are equivalent to WP3 CONNECTors and, hence, our synthesis algorithm supports WP3 CONNECTor synthesis. This answers **Q2**, as phrased above. To answer **Q1**, concerning the set of considered mediation patterns, our synthesis method relies on a revised version of the WP2 CONNECTor algebra. We recall that it is an algebra for reasoning about protocol mismatches where basic mismatches can be solved by suitably defined primitives, while complex mismatches can be settled by composition operators that build CONNECTors out of simpler ones. We revise the original algebra by adding an iterator operator and by giving its semantics in terms of the restricted and revised specification theory. Thus, the structure of a modular CONNECTor is expressed by means of the primitives of the revised algebra and its composition operators.

Beyond addressing the **Q1** and **Q2** reviewers' questions and ensuring *correctness* of the synthesized CONNECTors, a further contribution of our method is that it promotes CONNECTor *evolution*, hence easing the synthesis and maintenance of the CONNECTor's implementation code. Note that the support for CONNECTor evolution is another crucial aspect for the synthesis of CONNECTors. In IT12 (see Appendix A), we use a slightly revised version of a WP3 case study (borrowed from the work described in [13]) to illustrate that some changes can be applied on the synthesized modular CONNECTor by simply acting on its constituent mediators, without entirely re-synthesizing its protocol.

2.3.1 An informal overview on the automated synthesis of modular CONNECTors

In Figure 2.10, we show the main activities (as rounded-corner rectangles) and input/output artefacts (as text) respectively performed and manipulated by our method for the automated synthesis of modular CONNECTors. The numbers denote the order in which the activities are carried out.

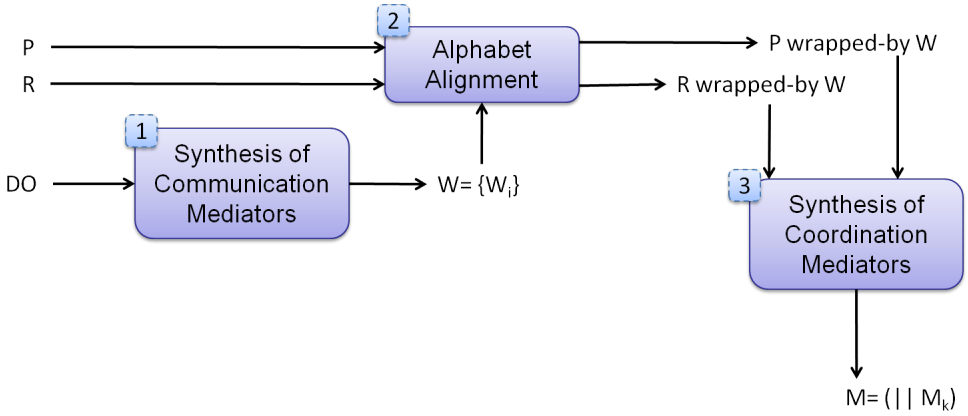


Figure 2.10: Overview of the automated synthesis of modular CONNECTors

We recall that a modular CONNECTor is a composition of independent mediators, each of them solving

a protocol mismatch. A mediator has an input-output behaviour (not necessarily strictly sequential, e.g., for allowing reordering of messages), and it is a "reactive" software entity harmonizing the interaction between heterogeneous NSs by intercepting output messages from one NS and eventually issuing to another NS the *co-related* input messages. Message co-relations can be inferred by taking into account ontological information.

Synthesis of communication mediators

The first activity of our method, "*Synthesis of Communication Mediators*", takes as input a domain ontology DO, for protocols P and R, and automatically synthesizes the set W of the so called *communication mediators*. Communication mediators are responsible for solving *communication mismatches*. These mismatches concern the semantics and granularity of protocol actions. For instance, by considering the domain of *purchase order systems*, it could be the case that P provides a single operation, e.g., $StartOrder$, to authenticate and create an empty order, whereas R expects to use two different operations, e.g., $Login$ and $CreateOrder$, one for the authentication and one for the creation of the empty order. As depicted in the figure, to solve these kind of mismatches it is necessary to assume and use ontology knowledge (DO) in order to align the two protocols to the same concepts and language.

Typically, ontologies account for two fundamental relations between concepts: *subsumption* and *aggregation* [4]. A concept a is *subsumed by* a concept b, in a given ontology O, if in every model of O the set denoted by a is a subset of the set denoted by b. A concept a is *an aggregate of* concepts $b_1, \ldots, b_n$ if the latter are part of the former. It is worth mentioning that our use of the ontology concept is specific to the CONNECT project. Thus, in the following, we will exploit these notions to our purposes. That is, concepts in DO correspond to NS input/output actions. The two relations between concepts are, then, used to account for the granularity of the data that define the structure of the messages exchanged by the respective input/output actions. For instance, by continuing the example introduced above, the input message (for P) associated to the request of $StartOrder$ is an aggregate of the output messages (for R) associated to the requests of $Login$ and $CreateOrder$. This means that the message associated to $StartOrder$ can be built by merging the messages associated to $Login$ and $CreateOrder$. Thus, a mediator would take $Login$ and $CreateOrder$ as input in any order from R, and send $StartOrder$ as the merge of $Login$ and $CreateOrder$ to P (plus possible additional data explicitly specified in DO for $StartOrder$). As introduced above, a revised version of the CONNECTor algebra is used to express the corresponding communication mediator as follows:

$$W_1 = [\![Trans(Login, x_1) \odot Trans(CreateOrder, x_2) \odot Prod(x_3) \odot Merge([x_1, x_2, x_3], StartOrder)]\!];$$

whose semantics is the interface automaton shown in Figure 2.11. It is obtained by giving semantics of the algebra primitives and operators through the restricted and revised version of the compositional specification theory, and by performing minimization with respect to hidden actions.

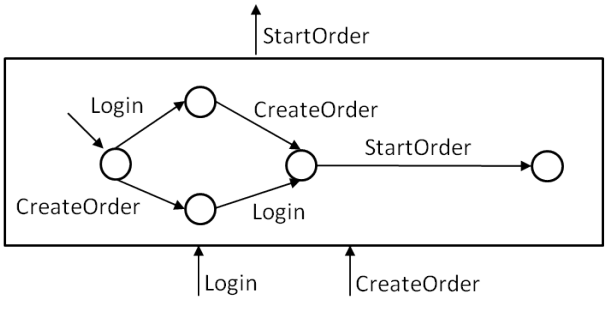


Figure 2.11: Interface automaton of the W_1 communication mediator

Subsumption ontological relationships are handled analogously. For instance, let us consider two new actions, $CloseOrder$ of R and $PlaceOrder$ of P. Furthermore, the message associated to $CloseOrder$ is subsumed by the message associated to $PlaceOrder$. This means that the set of data constituting $CloseOrder$ is a subset of those constituting $PlaceOrder$. Thus, in order to build the message associated

to $CloseOrder$, a mediator needs to process the data contained in the message associated to $PlaceOrder$ first. In other words, the mediator takes $PlaceOrder$ as input from P and sends, as co-related message, $CloseOrder$ as output to R. Note that, internally, this means consuming the extra data in $PlaceOrder$ that do not belong to set of data for $CloseOrder$. The corresponding communication mediator can be expressed as follows:

$$W_2 = [\![Split(PlaceOrder, [CloseOrder, x_2]) \odot Cons(x_2)]\!];$$

whose semantics is the interface automaton shown in Figure 2.12.

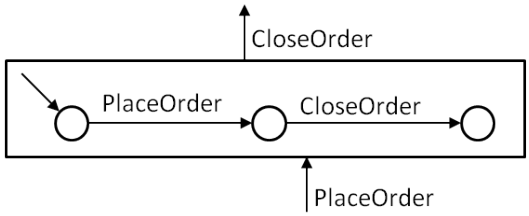


Figure 2.12: Interface automaton of the W_2 communication mediator

Alphabet alignment

Once the set of communication mediators is automatically synthesized, the second activity of our method, "*Alphabet Alignment*", starts by taking as input the two protocols, P and R, and the synthesized set, $\{W_i\}$, of communication mediators. By continuing our running example, for the sake of simplicity[3], let us consider a very simple protocol for P as shown in Figure 2.13.

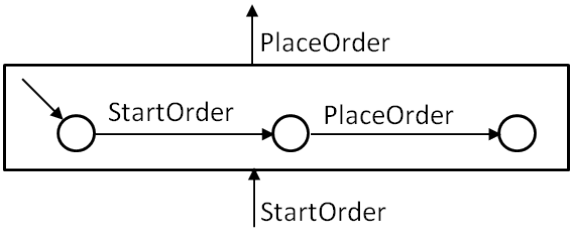


Figure 2.13: Interface automaton of P

Note that the alphabet of P is represented by the set of actions $\{StartOrder, PlaceOrder\}$. According to the discussion above, let us consider as alphabet for R the set $\{Login, CreateOrder, CloseOrder\}$. The aim of this activity is to exploit, as protocol wrappers, the synthesized communication mediators, W_1 and W_2, in order to "align" the alphabets of P and R, hence solving all the possible communication mismatches. Roughly speaking, the goal of this activity is to suitably exploit communication mediators in order to make two heterogeneous protocols "speak" the same language. In particular, when synthesized out of subsumption (resp., aggregation), a communication mediator is used as a wrapper for output (resp., input) actions of a protocol. Thus, we define a derived composition operator called *wrapping*. By continuing our running example, in the lowermost side of Figure 2.14, we show the interface automaton resulting from wrapping the interface automaton of P by the one of W_1.

Intuitively, as pictorially highlighted in Figure 2.14 by means of dashed arrows, the wrapping "fuses", into a single state, those transitions labeled by common actions (see the gray-colored actions in Figure 2.14) and their respective source and target states. Furthermore, each common action disappears in the result of the wrapping. In other words, the wrapping allows to achieve the purposes of the alphabet alignment activity since it allows to translate an action from an alphabet into a certain sequence of actions from another alphabet.

[3]By referring to Appendix A, the case study described in the paper IT12 considers a more complex protocol.

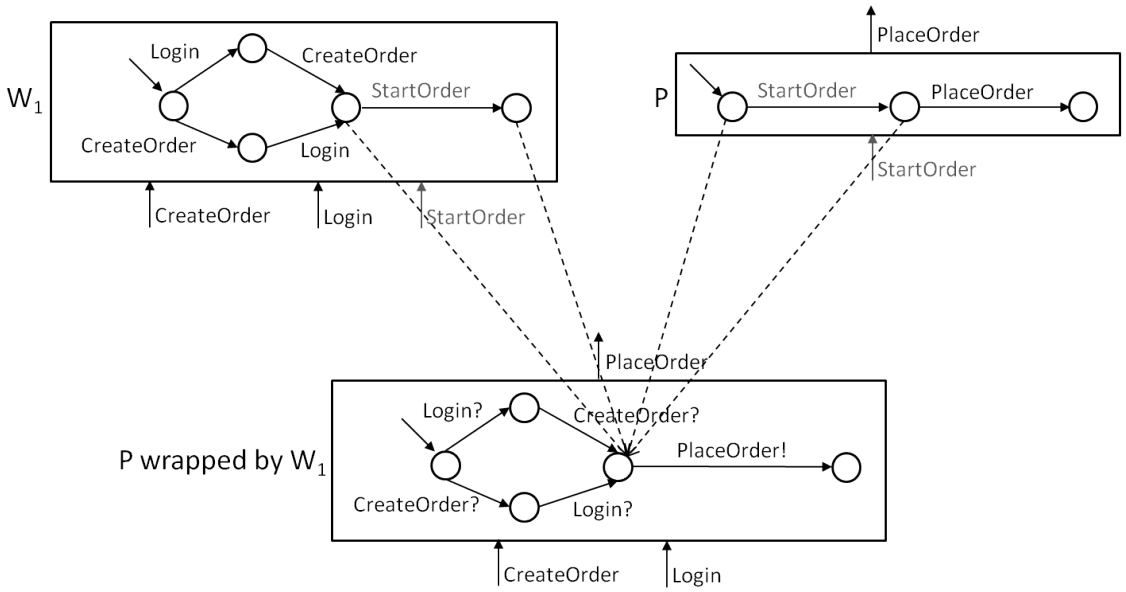


Figure 2.14: Interface automaton of W_1 (uppermost and left-hand side), P (uppermost and right-hand side), and "P wrapped by W_1" (lowermost side)

Going back to our example, in Figure 2.15 we show the interface automaton of P wrapped by W_1 and, in turn, wrapped by W_2.

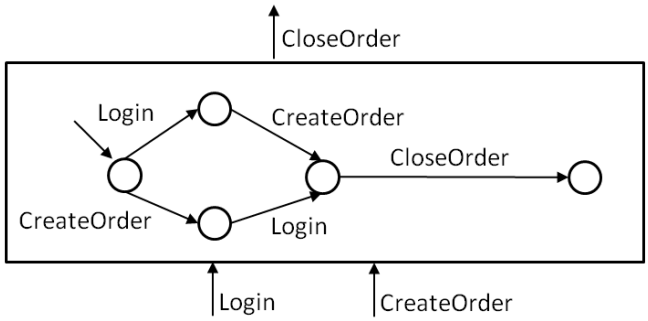


Figure 2.15: Interface automaton of P wrapped by W_1 and, in turn, wrapped by W_2

Thus, by exploiting the two synthesized communication mediators, W_1 and W_2, as wrappers for P, we made the alphabets of P and R the same. That is, at this stage, P and R speak the same language hence solving any communication mismatch. However, communication mediators are not able to solve all mismatches such as *coordination mismatches*. These mismatches deal with the control structure of the (wrapped) protocols. The last activity of our synthesis method is for solving such mismatches.

Synthesis of coordination mediators

Continuing our running example, to better illustrate the third activity of our method, let us consider a slightly extended version of both the "wrapped" version of P (hereafter, referred to simply as P) and R, whose IA are shown in Figure 2.16.

Although the two protocols shown in Figure 2.16 share the same alphabet of actions, their interaction can still exhibit some mismatches (i.e., coordination mismatches). They are due to (i) messages sent/received in a different order (see the sequences made of $ConfirmItem$ and $CloseOrder$); (ii) third-party messages ($PayThirdParty$); and (iii) extra/missing sends corresponding to redundant messages (possi-

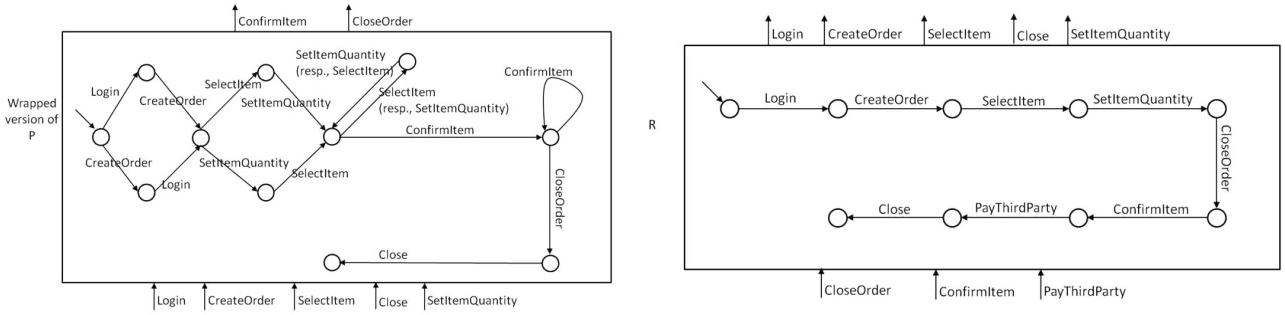


Figure 2.16: IA for P (left-hand side) and R (right-hand side)

bly also coming from looping/cyclic behavior, e.g., $SelectItem$ and $SetItemQuantity$). Thus, in general, the construction of other mediators that can delegate/receive[4], consume, produce, and reorder messages is required. We call these kinds of mediators *coordination mediators*.

The synthesis of coordination mediators is carried out by reasoning on the two sets of traces of P and R, denoted as $Tr(P)$ and $Tr(R)$, respectively. Note that P and R are prefix-closed and hence their sets of traces are finite. Furthermore, possible loops/cycles are considered k times (where k is a parameter of our synthesis algorithm). This means that our method produces finite sets of finite traces.

For all pairs of traces $(t_P, t_R) \in Tr(P) \times Tr(R)$, our method tries to synthesize a coordination mediator that makes the protocols corresponding to t_P and t_R able to interoperate. If no mediator has been synthesized, then a modular connector for P and R does not exist. Otherwise, a non-empty set of coordination mediators is produced. Indeed, considering all pairs in $Tr(P) \times Tr(R)$ is not needed. It is sufficient to consider only the subset of pairs of *semantically related* traces. Traces t_P and t_R are semantically related if every action that does not belong to their set of common actions is a third-party action, e.g., the action $PayThirdParty$ of P.

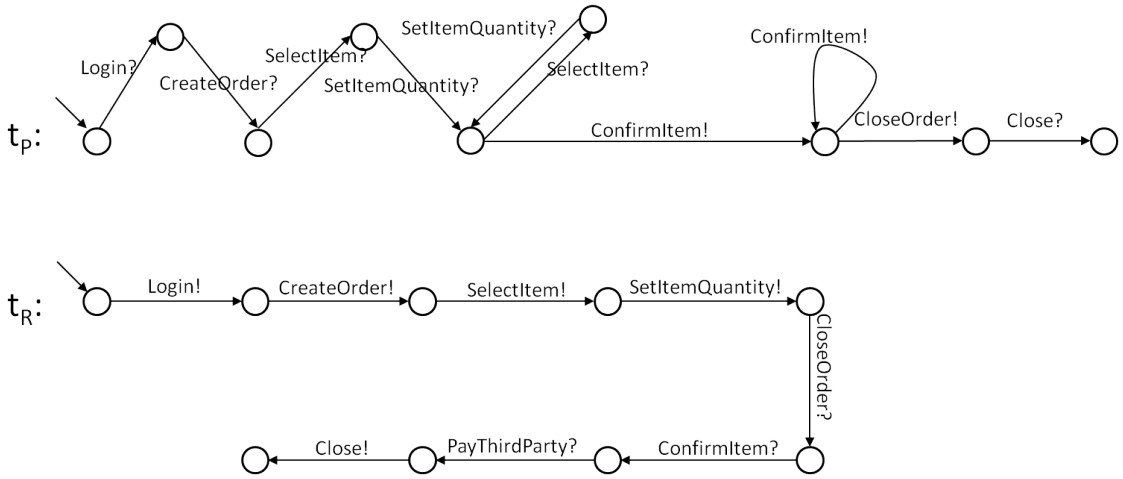


Figure 2.17: (t_P, t_R): a pair of semantically related traces for P and R

Figure 2.17 shows a pair of semantically related traces for P and R. Input (resp., output) actions are denoted by the question (resp., exclamation) mark. For each pair (t_P, t_R), the method computes the so called *difference pair* (t'_P, t'_R). Coming back to our example, Figure 2.18 shows the difference pair for the pair of traces shown in Figure 2.17.

t'_P (resp., t'_R) is a sub-trace of t_P (resp., t_R) representing, in a single sequence, the sequences of actions in which t_P (resp., t_R) differs from t_R (resp., t_P). Due to the alphabet alignment, finding a coordination mediator for t_P and t_R means finding a coordination mediator for t'_P and t'_R. As pictorially shown

[4]To/from a third-party.

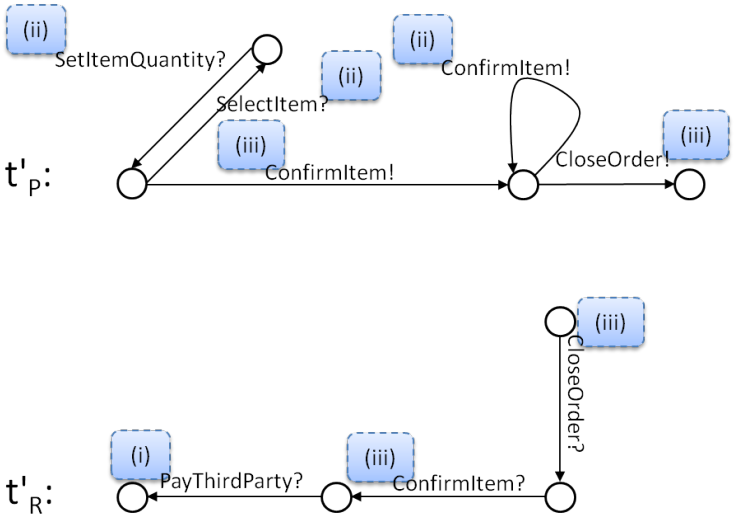


Figure 2.18: Difference pair for (t_P, t_R)

in Figure 2.18 by tagging actions with (i), (ii) or (iii), since t_P and t_R are semantically related and their loops/cycles are considered k times, t'_P and t'_R can be different for three reasons only: (i) they have unshared actions corresponding to input/output third-party actions; (ii) they exhibit extra/missing sends corresponding to redundant messages, possibly also coming from looping/cyclic behavior; and (iii) they have complementary shared actions that appear in a different order. By means of the coordination mediators to be synthesized, the first ones should be received by a third-party (resp., an NS) and delegated to the receiving NS (resp., third-party), the second ones should be produced/consumed, and the third ones should be reordered. Thus, by denoting with * the iterator operator (for details, see IT12 in Appendix A), the synthesized coordination mediators for our running example, only limited to t'_P and t'_R in Figure 2.18, are:

$$M_1 = [\![Trans(PayThirdParty, PayThirdParty')^*]\!];$$
$$M_2 = [\![Order([ConfirmItem, CloseOrder], (2,1), [ConfirmItem', CloseOrder'])^*]\!];$$
$$M_3 = [\![(Prod(SelectItem))^*]\!];$$
$$M_4 = [\![(Prod(SetItemQuantity))^*]\!];$$
$$M_5 = [\![(Cons(ConfirmItem))^*]\!].$$

The modular connector for our entire example is given by the following composition of coordination mediators: $M = M_1||\ldots||M_6$, where $M_6 = [\![Order([SelectItem, SetItemQuantity], (2,1), [SelectItem', SetItemQuantity'])^*]\!]$ and $||$ denotes the IA parallel composition operator (for details, see IT12 in Appendix A); plus the set $W = \{W_i\}$ of communication mediator used for the alphabet alignment. As formally shown in the next section, under alphabet alignment, M is a correct connector meaning that the CONNECTed system is free from communication and coordination mismatches, deadlocks only when each of P, M, and R deadlock, and satisfies the constraints imposed by the given domain ontology.

2.3.2 Evaluation: correctness and CONNECTor evolution

In this section, we evaluate our method for the automated synthesis of modular CONNECTors with respect to its support to: (i) WP3 CONNECTor synthesis; and (ii) CONNECTor evolution, which represents a crucial dimension for a highly dynamic scenario such as CONNECT.

As described in [12], the WP2 compositional specification theory can be used to synthesize, via a *quotient* operator $/$, a monolithic connector M such that $P||M||R$ *refines*[5] a given goal G, i.e., $M = G/(P||R)$. G can be considered as the *intent* of the CONNECTor to be synthesized or, equivalently, of the

[5]Under a suitable notion of refinement whose formal definition is given in IT12 (see Appendix A).

CONNECTed system to be. The formal definition of G is out of the scope of this work. For the purposes of this section, it is sufficient to say that G is an interface automaton, representing the CONNECTed system goal, which explicitly models three crucial conditions for correct communication and coordination: (*c1*) $P||M||R$ is not permitted to generate any inconsistencies; (*c2*) $P||M||R$ is only permitted to deadlock when all P, M, and R deadlock; and (*c3*) $P||M||R$ must satisfy the constraints imposed by the given protocol ontology.

As already mentioned above, CONNECTors synthesized through quotient generalizes WP3 CONNECTors. Thus, showing that our synthesis method supports the WP3 CONNECTor synthesis means to show that a modular CONNECTor enjoys the same correctness properties of a monolithic CONNECTor obtained via quotient, i.e., the *c1*, *c2*, and *c3* correctness conditions. This, in turn, means showing that a modular connector M synthesized for protocols P and R is such that *c1*, *c2*, and *c3* hold, under alphabet alignment. However, note that *c2* and *c3* trivially hold by construction. In fact, when composing in parallel protocols, the only possibility to have "sink" states concerns scenarios in which none of the protocols is willing to perform any action (*c2*); and communication mediators ensure alphabet alignment (*c3*). Thus, in this section, by considering W as the set of synthesized communication mediators, we focus on stating that the CONNECTed system made of:

- P wrapped by all mediators in W;
- M; and
- R wrapped by all mediators in W;

is free from inconsistencies (*c1*). To do this we can exploit our notion of refinement (see IT12 in Appendix A) to state the following theorem whose formal proof can be found in the related paper reported in Appendix A.

Theorem 2.3 (*Correctness under alphabet alignment*)
Let M be a modular connector synthesized for the aligned protocols "P wrapped by all mediators in W" and "R wrapped by all mediators in W", then the following properties hold: (1) *the* CONNECT*ed system made of M and "R wrapped by all mediators in W" refines any legal environment for "P wrapped by all mediators in W", and* (2) *the* CONNECT*ed system made of "P wrapped by all mediators in W" and M refines any legal environment for "R wrapped by all mediators in W".*

Concerning the ability, for modular connectors, to evolve in response of possible changes, the most interesting scenario is related to changes at the level of the domain ontology. In fact, syntactic changes at the level of the NSs' interface directly correspond to a relabeling of mediator inputs/outputs, and related concepts in the ontology. We recall that the synthesis of coordination mediators deals with sets of traces. Thus, changes at the protocol level imply to re-iter the synthesis step on the affected traces only, hence accordingly changing the corresponding mediators. However, in the worst case, i.e., all the traces of a protocol share at least one action, the entire synthesis step must be repeated.

As an example of a possible change at the level of the domain ontology, let us go back to our running example and apply the following modification to the domain ontology: remove the ontological constraint for which $StartOrder$ is an aggregate of $Login$ and $CreateOrder$, and add the two constraints for which $StartOrder$ is subsumed by both $SelectItem$ and $SetItemQuantity$. Although simple, this change highlights the effectiveness of our decomposition with respect to supporting CONNECTor evolution. In fact, to address the applied change, it is sufficient to reason compositionally at the level of the algebra-based description of the modular connector M and related set W of communication mediators, instead of reasoning in terms of its underlying IA-based monolithic representation. In particular, by just looking at the mediators' interface, one can easily recognize that the communication mediator affected by the proposed change is W_1, while no coordination mediator is affected. Due to the fact that the above mentioned aggregation constraint has been removed from the domain ontology, W_1 is removed as well. In place of it, two communication mediators, $W_{1.1} = [\![(Split(SelectItem, [StartOrder, z]) \odot Cons(z))^*]\!]$ and $W_{1.2} = [\![(Split(SetItemQuantity, [StartOrder, k]) \odot Cons(k))^*]\!]$, are synthesized due to the addition in the ontology of the two above considered subsumption constraints. Furthermore, we recall that the interface automaton of P has been modified in order to align its alphabet to the one of R.

To reflect the change on the performed alphabet alignment, a trace in $Tr(R)$ that contains $SelectItem$ and/or $SetItemQuantity$ is modified by considering the following substitution: $\{StartOrder/SelectItem, StartOrder/SetItemQuantity\}$. Analogously, a trace in $Tr(P)$ that contains either the sequence $\langle SelectItem\ SetItemQuantity\rangle$ or $\langle SetItemQuantity\ SelectItem\rangle$ is modified by replacing any of these sequences with $StartOrder$. According to the new alphabet alignment, in place of both M_3 and M_4 the coordination mediator $[\![(Prod(StartOrder))^*]\!]$ is synthesized. Note that, in the monolithic connector, $SelectItem$ and $SetItemQuantity$ would always appear one after the other and modifying the connector according to the applied change would mean to solve again the entire quotient problem.

3 Evaluation of WP2 results

In this chapter, we report an overall evaluation of the WP2 results with respect to the objectives and assessment criteria discussed in Deliverable D6.3 [14], hence refining the preliminary evaluation contained in that deliverable. We recall that the objective of WP2 is to provide a comprehensive theory to enable composition and automated synthesis of CONNECTors in order to automatically learn and reason about CONNECTor behaviours via a quantitative assume-guarantee reasoning paradigm. The expected outcomes are theories, formalisms, and methods that can be used for the specification, design, development, and automated synthesis of CONNECTors, allowing for both functional and non-functional properties to be expressed and verified. Throughout the duration of the project, we have developed quantitative assume-guarantee reasoning techniques able to express and manage both functional and non-functional properties of CONNECTors. We have also developed compositional techniques for enabling automated synthesis of CONNECTors whose modularity supports dynamic evolution. All of the achieved results have been published in leading international conferences. The success of this work package will be evaluated on the final specification theory formalism, along with the defined method for the automated synthesis of modular CONNECTors. There are two criteria for the evaluation.

Objective 1: Innovative formalism

- *Assessment criterion:* since there are many formalisms in the literature that address system modelling and synthesis, we will assess our formalisms/methods on their significance, in the sense of advancing the state of the art, and the range of functionality of the operators that are supported. This can be judged by high quality publications, e.g., in leading international conferences and journals.
- *Methodology:* the evaluation of our formalisms/methods will be based on assessment against other state of the art formulations, in addition to our related publication acceptance at leading venues.
- *Assessment:* the main contributions of the project are compositional quantitative assume-guarantee reasoning and a compositional specification theory for components. Our results were published in internationally leading conferences, and we have a number of journal papers under submission or in preparation. Among the leading conferences listed in [10] - overall 25 conferences, which means more than 7 papers published per year - it is worth mentioning FACS2012, TACAS2010, TACAS2011, FASE2011, ESEC-FSE2009, ICSE2009, FORMATS2009, FORMATS2010, FORMATS2012, QEST2011, and QEST2012.

  We proposed the first compositional assume-guarantee framework for probabilistic systems, both for safety as well as liveness properties (papers accepted for the leading conferences TACAS2010 and TACAS2011). We formulated and implemented model checking algorithms based on multi-objective model checking, demonstrating encouraging performance. We also extended the compositional framework with automated assumption learning for a subset of rules (reported in QEST2010 and FASE2011).

  Our specification theory addresses a number of shortcomings of interface automata and supports a broader range of operators than previously achieved. In particular, we propose the first quotient for non-deterministic interface automata, the first definition of conjunction on this model type, and the weakest preorder preserving substitutivity. Furthermore, the developed real-time extension of the specification theory allows the modelling of the real-time performance of networked systems. From this, as a further advance on the state of the art, we have formulated a technique based on quotient for synthesising connectors that are not only functionally compatible with their environment, but that are also responsive to the timing demands of the systems to be made interoperable. The first paper about the specification theory was accepted to ESOP2012, a leading symposium on programming, followed by the timed extension (FORMATS2012) and the assume-guarantee reasoning framework (FACS2012).

  The defined method for the automated synthesis of modular CONNECTors represents a significant progress with respect to state of the art methods, where the output of the synthesis process results in a monolithic connector that prevents evolution, and makes synthesis and maintenance of the connector code a difficult task. The significance of our method has been confirmed by acceptance

of a related publication at ICSE2013, a leading conference with interest in system modelling and reasoning.

It is worth mentioning that a focused amount of attention has been given to component-based specification theories recently, by numerous authors, justifying that our work has tangible benefits, both in terms of its practical applicability, and technical contribution (cf [5, 6]). Other work has focused on components with different semantics, such as those based on modalities [20, 28].

Objective 2: Introducing the theoretical underpinning for emergent connectors

- *Assessment criterion:* WP2 intends to provide the theoretical underpinning for the work carried out in the other work packages, in the sense that **connectors specified in WP2 can be instantiated** in WP3 (synthesis), WP4 (learning) and WP5 (dependability analysis). Thus a key challenge when developing the specification theory and its related synthesis methods (e.g., quotient-based, for producing modular CONNECTors) is to ensure that it is fully integrated and usable by WPs 3, 4 and 5.
- *Methodology:* we will assess the application of the formalisms/methods to the concrete approaches developed in WPs 3-5.
- *Assessment:* we have studied CONNECT-relevant examples where they have been available, for example, mediator synthesis from WP3. In particular, as described in this deliverable, we have developed an example based on the GMES scenario, for which we synthesise connectors in order to allow successful communication between a number of networked systems. This example has been applied to both the non-quantitative assume-guarantee framework for safety properties, as well as the real-time extension of the specification theory. We have also developed an extension of the specification theory in D2.3 [12] aimed at the register automata of WP4, but without imposing restrictions (determinism, canonicity) where this is not necessary. We have applied the quantitative assume-guarantee verification framework to the dependability analysis in WP5 (see D2.2 [11] and D2.3).

  We have demonstrated integration with WP3 in both D2.3 and this deliverable. In particular, in D2.3, we have shown how quotient can be used for synthesis by showing that a mediator synthesised by WP3 is the most general mediator, and that the theory supports checking for freedom of errors. As a further proof of integration, in this deliverable, we have shown how to integrate the specification theory with the WP2 connector algebra so as to support CONNECTor synthesis in WP3. In particular, we have provided a method for the automated synthesis of modular CONNECTors that enjoy the same correctness properties of the monolithic CONNECTor obtained by expressing the synthesis problem as a quotient problem, as specified by the specification theory and as it is equivalently produced by the WP3 synthesis. Moreover, to show how modular CONNECTors support evolution, we applied our method on a WP3 case study in the e-commerce domain borrowed from D3.3 [13].

4 Conclusion

In this deliverable, we reported the progress of WP2 during the fourth year of the CONNECT project. The work has been organized into two main streams: (i) a quantitative extension of the compositional specification theory devised during the previous two years; and (ii) the integration of the WP2 CONNECTor algebra with the specification theory so as to support WP3 CONNECTor synthesis. As main outcomes of WP2 for the fourth year of the project, these two streams of work have led us to define an *AG framework for safety properties*, a *quantitative specification theory*, and an automated *method for the synthesis of modular* CONNECT*ors*. In the following, for each outcome, we report a brief discussion on groundbreaking research challenges solved and open.

- **AG framework for safety properties.** We have presented an algebraically elegant framework for reasoning compositionally about the safety properties satisfied by specifications of components. Our work is more general than existing attempts in the area [24], as we do not insist on equality of interfaces under satisfaction, do not require components and guarantees to be input-enabled, provide the first definition of conjunction and quotient on contracts for this model type, and present strong algebraic properties for the operations defined directly on contracts.

  We are currently looking at a liveness extension of the framework, based on quiescence (in order to stay within the present setup of finite traces). To facilitate this, besides assumptions and guarantees on contracts, we also include a set of liveness traces on which implementing components may not become quiescent (i.e., they must be able to extend such a trace by an output, without further stimulation from the environment). This work has also resulted in a quiescence extension of the compositional specification theory, where refinement guarantees both substitutivity and progress, the latter being achieved by requiring that a quiescent trace in a refining component must have been quiescent in the original. Present work involves submission of these two frameworks.

- **Quantitative extension of the specification theory.** This framework supports the modelling of real-time components with critical timing constraints. As for the non-quantitative specification theory, we define the full collection of compositional operators, including quotient for synthesis, and provide the weakest substitutive refinement preorder maintaining absence of safety and bounded liveness errors. Our theory is elegant in comparison to existing approaches, based on our equating of safety and liveness errors through a single inconsistent state $\perp$. This means that we need only use a single transition system, rather than two (cf [19]).

  Subsequent work has led us to look at the realisable sub class of timed components, which are not permitted to stop the global system clock. This involves redefining the compositional operators, in order to avoid encountering a timestop state from which time may not progress. In addition to this, we also plan to look at developing an AG framework for reasoning compositionally about timed components. This will strengthen the parallels between our non-quantitative theories and this timed extension, as well as our eventual goal of formulating a specification theory for models exhibiting both real-time and discrete probabilistic behaviour (work in progress).

  The specification theory developed as part of WP2 is based on linear-time refinement (to align with automata learning in WP4) and supports operations such as conjunction for independent development and quotient for connector synthesis (to align with WP3). In the last reporting period, we focused on developing assume-guarantee rules for our specification theory as well as a quantitative, timed extension of the specification theory, which aligns with the monitoring framework (WP5). One additional advantage of the timed extension is that refinement can be defined in terms of timed trace containment, a natural extension of the refinement defined in the non-quantitative case. In contrast, probabilistic refinement defined in terms of traces is known not to be compositional, and is usually based on simulation relations. As a first step, we have formulated a refinement relation for abstract probabilistic timed automata based on modal specifications, together with parallel and conjunction (work in progress, not reported here). Future work will include formulation of quotient as well as linking the results to the quantitative, probabilistic assume-guarantee framework developed in earlier deliverables that supports a parallel operator and a number of compositional reasoning rules.

- **Automated synthesis of modular CONNECTors.** A modular connector is synthesized as a composition of independent mediators, each of them corresponding to the solution of a recurring protocol mismatch. We have proven that our connector decomposition is correct and, by means of a WP3 case study, we have shown how it promotes connector evolution. An overall advantage of our approach with respect to the work in the state of the art (see [32, 29, 25, 22, 30, 27, 8] just to mention a few) is that our connectors have a modular software architecture organized as a composition of fundamental mediation primitives. This supports connector evolution and automated generation of the connector's implementation code. In particular, we have recently released a first implementation (http://code.google.com/p/otf-connector/) of both the algebra primitives and the plugging operator. This implementation is based on the use of Enterprise Integration Patterns (http://www.eaipatterns.com/) and is developed through the Apache Camel framework (http://camel.apache.org/). Because of the way a modular connector is structured, the automatic generation of its actual code written in terms of our algebra implementation is viable and can be achieved with little effort. We have started to show, through its application to the real world case study presented in IT12 (see Appendix A), that our method supports connector evolution.

  As future work, we intend to carry out a rigorous empirical investigation to confirm the results reported in IT12. Another future research direction concerns the ability to infer the needed ontological information, out of the interface description of the two protocols, rather than assuming it as given.

A Published work

In this appendix, we report some CONNECT papers that have been accepted for publication to leading international conferences during the fourth year of the project. These papers provide details on the work undertaken by WP2, whose overview has been given in this deliverable. The reported papers have been attached in the same order as listed below.

CJK12 C. Chilton, B. Jonsson, and M. Kwiatkowska. *Assume-Guarantee Reasoning for Safe Component Behaviours*. In C. Pasareanu and G. Salaun, editors, Proc. 9th International Symposium on Formal Aspects of Component Software (FACS'12), Lecture Notes in Computer Science, vol. 7684, pp. 92-109. Springer-Verlag, 2012.

The relation of this paper to CONNECT is documented in Section 2.1.

CKW12 C. Chilton, M. Kwiatkowska, and X. Wang. *Revisiting Timed Specification Theories: A Linear-Time Perspective*. In M. Jurdzinski and D. Nickovic, editors, Proc. 10th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'12), Lecture Notes in Computer Science, vol. 7595, pp. 75-90. Springer-Verlag, 2012.

An exposition of this paper is contained in Section 2.2.

IT12 *Automatic Synthesis of Modular Connectors via Composition of Protocol Mediation Patterns*. In Proc. of the 35th International Conference on Software Engineering (ICSE 2013). To appear.

It provides details for the work discussed in Section 2.3.

A.1 Assume-Guarantee Reasoning for Safe Component Behaviours

Assume-Guarantee Reasoning for Safe Component Behaviours

Chris Chilton[1], Bengt Jonsson[2], and Marta Kwiatkowska[1]

[1] Department of Computer Science, University of Oxford, UK
[2] Department of Information Technology, Uppsala University, Sweden

Abstract. We formulate a sound and complete assume-guarantee framework for reasoning compositionally about safety properties of component behaviours. The specification of a component, which constrains the temporal ordering of input and output interactions with the environment, is expressed in terms of two prefix-closed sets of traces: an assumption and guarantee. The framework supports dynamic reasoning about components and specifications, and includes rules for parallel composition, logical conjunction corresponding to independent development, and quotient for incremental synthesis. Practical applicability of the framework is demonstrated by considering a simple printing example.

Keywords: assume-guarantee, specification theory, components, compositionality, parallel, conjunction, quotient.

1 Introduction

Component-based design methodologies enable both design- and run-time assembly of software systems from heterogeneous components, thus facilitating component reuse, incremental development and independent implementability. To improve the reliability and predictability of such systems, specification theories have been proposed that permit the mixing of specifications and implementations, and allow for the construction of new components from existing ones by means of compositional operators [1,2,3]. A specification should make explicit the *assumptions* that a component can make about the environment, and the corresponding *guarantees* that it will provide about its own behaviour. This allows for the use of compositional assume-guarantee (AG) reasoning, in order to combat issues of complexity and state space explosion during system development and verification.

In earlier work [4], we introduced a component-based specification theory, in which components communicate by synchronisation of I/O actions, with the understanding that inputs are controlled by the environment, while outputs (which are non-blocking) are under the control of the component. The component-model is conceptually similar to the interface automata of de Alfaro and Henzinger [5], except that our refinement is based on classical sets of traces, as opposed to alternating simulation, and that we allow explicit specification of inconsistent traces, which can model underspecification and errors, etc. With both trace-based and

operational representations for components, a distinguishing feature of our theory is the inclusion of conjunction and quotient operators (which generalise those of [2,6]) for supporting independent and incremental development, respectively. Logical disjunction and hiding can also be added. The theory enjoys strong algebraic properties with all the operators being compositional under refinement, and we prove full abstraction with respect to a simple testing framework.

In [4] and [5], the assumptions and guarantees of components are merged into one behavioural representation. In many cases, this avoids duplication of common information, although there are situations in which it is desirable to manipulate the assumptions and guarantees separately. For instance, we may want to express a simple guarantee (such as "no failure will occur") without having to weave it into a complex assumption. Another advantage of separation is specification reuse, in that the same guarantees (or assumptions) can be used for several related interfaces, each representing different versions of a component.

Contributions. In this paper, we present a complete specification theory for reasoning about AG specifications of components (as modelled in [4]). Assumptions and guarantees are prefix-closed sets of traces, meaning our framework facilitates reasoning about safety behaviours, and differs from (arguably) more complex approaches based on modal specifications and alternating simulation. Building upon the theory in [4], we define the operators of parallel, conjunction and quotient directly on AG specifications (the last being the first such definition), and prove their compositionality. By treating AG specifications as first-class citizens, the theory supports flexible development and verification of component-based systems using AG principles. A component can be characterised by its weakest AG specification, and, in the opposite direction, we can infer the least refined component satisfying a given specification. From this, a notion of refinement corresponding to implementation containment is defined. In relating implementations with AG specifications by means of satisfaction, we formulate a collection of sound and complete AG reasoning rules for the preservation of safety properties under the operations and refinement preorder of the specification theory. These rules are inspired by the Compositionality Principle of [7,8] for parallel composition, which we generalise to the operations of conjunction and quotient. The rules allow us to infer properties of compositions for both AG specifications and components, thus enabling designers to deduce whether it is safe to substitute a component, for example one synthesised at run-time by means of the quotient operator, with another.

Related Work. Compositional AG reasoning has been extensively studied in the literature, where traditionally the work was concerned with compositional reasoning for processes, components and properties expressed in temporal logics [9,10,11]. A variety of rule formats have been proposed, although Maier demonstrates through a set-theoretic setting in [12] that compositional circular AG rules for parallel composition (corresponding to intersection) cannot both be sound and complete. This seems to contradict the work of Namjoshi and Tre-

fler [13], although the discrepancy is attributed to the fact that the sound and complete circular rule presented in [13] is non-compositional.

Compositional reasoning about AG specifications in the form of AG pairs, similar to what we consider in this paper, is discussed in [7] for the generic setting of state-based processes. The authors formulate a Compositionality Principle for parallel composition, and observe that this is sound for safety properties. A logical formulation for specifications is then discussed in [8], where intuitionistic and linear logic approaches are put forward. The main difference with our approach is that we consider an action-based component model and have a richer set of composition operators, including conjunction and quotient. We also prove completeness, by relying on the convention that an output is controlled by at most one component, which can be used to break circularity.

More recent proposals focus on compositional verification for interface theories [14,15], namely interface and I/O automata, which are closest to our work. In [14], Emmi *et al.* extend a learning-based compositional AG method to interface automata. They only consider the much more limited asymmetric rules for safety properties, which are shown to be both sound and complete. The rules are supplied for the original subset of operators and relations defined in [5], namely compatibility, parallel composition and refinement based on alternating simulation. Thus, no consideration is given to conjunction or quotient. Other notable work concerning compositional reasoning for interface theories is the AG framework defined by Larsen *et al.* in [15] for I/O automata, where assumptions and guarantees are themselves specified as I/O automata. The authors consider a parallel composition operator on AG specifications that is the weakest specification for composed components respecting independent implementability, for which they present a sound and complete rule. Our work allows a more general component model that does not require input-enabledness, and allows for specifications to have non-identical interfaces to their implementations. We go beyond [15] by defining conjunction and quotient operations directly on AG specifications, thus providing a significantly richer basis for AG based reasoning and development, and we do not require input-enabledness of guarantees.

A compositional specification theory based on modal specifications has been developed in [3], which includes all the operations we consider in this paper, but for systems without I/O distinction. Larsen *et al.* consider a cross between modal specifications and interface automata [1], where refinement is given in terms of alternating simulation/modal refinement (which is stronger than our trace containment), and no operations for conjunction and quotient are given. Surveying [16], Bauer *et al.* provide a generic construction for obtaining a contract framework based on AG pairs from a component-based specification theory. The abstract ideas share similarity with our framework, and it is interesting to note how parallel composition of contracts is defined in terms of the conjunction and quotient operators of the specification theory. Our work differs in that we define both of these operators directly on contracts. A definition of conjunction on contracts is provided in [17], but this is for a simplified contract framework, as witnessed by the definition of parallel composition on contracts.

Outline. In Section 2 we summarise the compositional specification theory of [4], which serves as a basis for our AG reasoning framework. Section 3 introduces the main definitions of the AG framework, and presents a number of sound and complete compositional rules for the operators of the specification theory. An application of our framework is illustrated in Section 4, while Section 5 concludes our work and suggests possible extensions. Proofs of our results are made available as the technical report [18].

2 Compositional Specification Theory

In this section, we briefly survey the essential features of our compositional specification theory presented in [4]. In that paper, we present two notations for modelling components: a trace-based formalism and an operational representation. Here we focus on the trace-based models, since operational models can be mapped to semantically equivalent trace-based ones.

A component comes equipped with an interface, together with a set of behaviours over the interface. The interface is represented by a set of input actions and a set of output actions, which are necessarily disjoint, while the behaviour is characterised by sets of traces.

Definition 1 (Components). *A* component $\mathcal{P}$ *is a tuple* $\langle \mathcal{A}^I_\mathcal{P}, \mathcal{A}^O_\mathcal{P}, T_\mathcal{P}, F_\mathcal{P} \rangle$ *in which* $\mathcal{A}^I_\mathcal{P}$ *and* $\mathcal{A}^O_\mathcal{P}$ *are disjoint sets referred to as inputs and outputs respectively (the union of which is denoted by* $\mathcal{A}_\mathcal{P}$*),* $T_\mathcal{P} \subseteq \mathcal{A}^*_\mathcal{P}$ *is a non-empty set of permissible traces, and* $F_\mathcal{P} \subseteq \mathcal{A}^*_\mathcal{P}$ *is a set of inconsistent traces. The trace sets must satisfy the constraints:*

1. $F_\mathcal{P} \subseteq T_\mathcal{P}$
2. *If* $t \in T_\mathcal{P}$ *and* $i \in \mathcal{A}^I_\mathcal{P}$*, then* $ti \in T_\mathcal{P}$
3. $T_\mathcal{P}$ *is prefix closed*
4. *If* $t \in F_\mathcal{P}$ *and* $t' \in \mathcal{A}^*_\mathcal{P}$*, then* $tt' \in F_\mathcal{P}$*.*

The permissible traces contain all possible interaction sequences between the component and the environment; they are thus receptive to all inputs, as these are under the control of the environment. If on some interaction sequence an error arises in the component, or the environment issues a non-enabled input, the trace is said to be inconsistent. We adopt the convention that any inconsistent trace is suffix closed, meaning that, once the component becomes inconsistent, it behaves similarly to the process CHAOS in CSP.

From hereon let $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{R}$ be components with signatures $\langle \mathcal{A}^I_\mathcal{P}, \mathcal{A}^O_\mathcal{P}, T_\mathcal{P}, F_\mathcal{P} \rangle$, $\langle \mathcal{A}^I_\mathcal{Q}, \mathcal{A}^O_\mathcal{Q}, T_\mathcal{Q}, F_\mathcal{Q} \rangle$ and $\langle \mathcal{A}^I_\mathcal{R}, \mathcal{A}^O_\mathcal{R}, T_\mathcal{R}, F_\mathcal{R} \rangle$ respectively.

Notation. Let $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ be sets of actions. For a trace t, write $t \upharpoonright \mathcal{A}$ for the projection of t onto $\mathcal{A}$. Now for $T \subseteq \mathcal{A}^*$, write $T \upharpoonright \mathcal{B}$ for $\{t \upharpoonright \mathcal{B} : t \in T\}$, $T \Uparrow \mathcal{B}$ for $\{t \in \mathcal{B}^* : t \upharpoonright \mathcal{A} \in T\}$, $T \Uparrow\!\Uparrow \mathcal{B}$ for $\epsilon + (T \Uparrow \mathcal{B})(\epsilon + \mathcal{A}^I)$, $T \uparrow \mathcal{B}$ for $T(\mathcal{B})(\mathcal{A} \cup \mathcal{B})^*$, $T \uparrow_\epsilon \mathcal{B}$ for $T \cup (T \uparrow \mathcal{B})$, $\overline{T}$ for $\mathcal{A}^* \setminus T$, and $\mathsf{pre}(T)$ for the largest prefix-closed set contained in T.

Refinement. In the specification theory, refinement corresponds to safe-substitutivity. This means that $\mathcal{Q}$ is a refinement of $\mathcal{P}$ if $\mathcal{Q}$ can be used safely in any environment that is safe for $\mathcal{P}$. An environment is safe for a component if any interaction between the two cannot be extended by a sequence of output actions under the control of the component such that the resulting trace is inconsistent. We will thus need to consider the safe representation of a component, obtained by propagating inconsistencies backwards over outputs.

Definition 2 (Safe component). *Let $\mathcal{P}$ be a component. The most general safe representation for $\mathcal{P}$ is a component $\mathcal{E}(\mathcal{P}) = \langle \mathcal{A}^I_\mathcal{P}, \mathcal{A}^O_\mathcal{P}, T_{\mathcal{E}(\mathcal{P})}, F_{\mathcal{E}(\mathcal{P})} \rangle$, where $T_{\mathcal{E}(\mathcal{P})} = T_\mathcal{P} \cup F_{\mathcal{E}(\mathcal{P})}$ and $F_{\mathcal{E}(\mathcal{P})} = \{tt' \in \mathcal{A}^*_\mathcal{P} : t \in T_\mathcal{P}$ and $\exists t'' \in (\mathcal{A}^O_\mathcal{P})^* \cdot tt'' \in F_\mathcal{P}\}$.*

We can now give the formal definition of refinement. Intuitively, $\mathcal{Q}$ must be willing to accept any input that $\mathcal{P}$ can accept, but it must produce no more outputs than $\mathcal{P}$, otherwise we could not be certain how the environment would respond to these additional outputs.

Definition 3 (Refinement). *For components $\mathcal{P}$ and $\mathcal{Q}$, $\mathcal{Q}$ is said to be a refinement of $\mathcal{P}$, written $\mathcal{Q} \sqsubseteq_{imp} \mathcal{P}$, iff:*

1. $\mathcal{A}^I_\mathcal{P} \subseteq \mathcal{A}^I_\mathcal{Q}$
2. $\mathcal{A}^O_\mathcal{Q} \subseteq \mathcal{A}^O_\mathcal{P}$
3. $T_{\mathcal{E}(\mathcal{Q})} \subseteq T_{\mathcal{E}(\mathcal{P})} \cup T_{\mathcal{E}(\mathcal{P})} \uparrow (\mathcal{A}^I_\mathcal{Q} \setminus \mathcal{A}^I_\mathcal{P})$
4. $F_{\mathcal{E}(\mathcal{Q})} \subseteq F_{\mathcal{E}(\mathcal{P})} \cup T_{\mathcal{E}(\mathcal{P})} \uparrow (\mathcal{A}^I_\mathcal{Q} \setminus \mathcal{A}^I_\mathcal{P})$.

The set $T_{\mathcal{E}(\mathcal{P})} \uparrow (\mathcal{A}^I_\mathcal{Q} \setminus \mathcal{A}^I_\mathcal{P})$ represents the extension of $\mathcal{P}$'s interface to include all inputs in $\mathcal{A}^I_\mathcal{Q} \setminus \mathcal{A}^I_\mathcal{P}$. As these inputs are not ordinarily accepted by $\mathcal{P}$, they are treated as bad inputs, hence the suffix closure with arbitrary behaviour.

Parallel Composition. The parallel composition of two components is obtained as the cross-product by synchronising on common actions and interleaving on independent actions. To support broadcasting, we make the assumption that inputs and outputs synchronise to produce outputs. Communication mismatches arising through non-input enabledness automatically appear as inconsistent traces in the product, on account of our component formulation. As the outputs of a component are controlled locally, we assume that the output actions of the components to be composed are disjoint.

Definition 4 (Parallel composition). *Let $\mathcal{P}$ and $\mathcal{Q}$ be components such that $\mathcal{A}^O_\mathcal{P} \cap \mathcal{A}^O_\mathcal{Q} = \emptyset$. Then $\mathcal{P} \,||\, \mathcal{Q}$ is the component $\langle \mathcal{A}^I_{\mathcal{P}||\mathcal{Q}}, \mathcal{A}^O_{\mathcal{P}||\mathcal{Q}}, T_{\mathcal{P}||\mathcal{Q}}, F_{\mathcal{P}||\mathcal{Q}} \rangle$, where:*

- $\mathcal{A}^I_{\mathcal{P}||\mathcal{Q}} = (\mathcal{A}^I_\mathcal{P} \cup \mathcal{A}^I_\mathcal{Q}) \setminus (\mathcal{A}^O_\mathcal{P} \cup \mathcal{A}^O_\mathcal{Q})$
- $\mathcal{A}^O_{\mathcal{P}||\mathcal{Q}} = \mathcal{A}^O_\mathcal{P} \cup \mathcal{A}^O_\mathcal{Q}$
- $T_{\mathcal{P}||\mathcal{Q}} = [(T_\mathcal{P} \Uparrow \mathcal{A}_{\mathcal{P}||\mathcal{Q}}) \cap (T_\mathcal{Q} \Uparrow \mathcal{A}_{\mathcal{P}||\mathcal{Q}})] \cup F_{\mathcal{P}||\mathcal{Q}}$
- $F_{\mathcal{P}||\mathcal{Q}} = [(T_\mathcal{P} \Uparrow \mathcal{A}_{\mathcal{P}||\mathcal{Q}}) \cap (F_\mathcal{Q} \Uparrow \mathcal{A}_{\mathcal{P}||\mathcal{Q}})]\mathcal{A}^*_{\mathcal{P}||\mathcal{Q}} \cup$
  $[(F_\mathcal{P} \Uparrow \mathcal{A}_{\mathcal{P}||\mathcal{Q}}) \cap (T_\mathcal{Q} \Uparrow \mathcal{A}_{\mathcal{P}||\mathcal{Q}})]\mathcal{A}^*_{\mathcal{P}||\mathcal{Q}}$.

Informally, a trace is permissible in $\mathcal{P} \parallel \mathcal{Q}$ if its projection onto $\mathcal{A}_\mathcal{P}$ is a trace of $\mathcal{P}$ and its projection onto $\mathcal{A}_\mathcal{Q}$ is a trace of $\mathcal{Q}$. A trace is inconsistent if it has a prefix whose projection onto the alphabet of one of the components is inconsistent and the projection onto the alphabet of the other component is a permissible trace of that component.

Conjunction. The conjunction of components $\mathcal{P}$ and $\mathcal{Q}$ is the coarsest component that will work safely in any environment that $\mathcal{P}$ or $\mathcal{Q}$ can work safely in. It can be thought of as finding a common implementation for a number of specifications. Thus, conjunction is essentially the meet operator on the refinement preorder. Consequently, the conjunction of two components is only defined when the union of their inputs is disjoint from the union of their outputs.

Definition 5 (Conjunction). *Let $\mathcal{P}$ and $\mathcal{Q}$ be components such that $\mathcal{A}^I_\mathcal{P} \cup \mathcal{A}^I_\mathcal{Q}$ and $\mathcal{A}^O_\mathcal{P} \cup \mathcal{A}^O_\mathcal{Q}$ are disjoint. Then $\mathcal{P} \wedge \mathcal{Q}$ is the component $\langle \mathcal{A}^I_{\mathcal{P}\wedge\mathcal{Q}}, \mathcal{A}^O_{\mathcal{P}\wedge\mathcal{Q}}, T_{\mathcal{P}\wedge\mathcal{Q}}, F_{\mathcal{P}\wedge\mathcal{Q}}\rangle$, where:*

- $\mathcal{A}^I_{\mathcal{P}\wedge\mathcal{Q}} = \mathcal{A}^I_\mathcal{P} \cup \mathcal{A}^I_\mathcal{Q}$
- $\mathcal{A}^O_{\mathcal{P}\wedge\mathcal{Q}} = \mathcal{A}^O_\mathcal{P} \cap \mathcal{A}^O_\mathcal{Q}$
- $T_{\mathcal{P}\wedge\mathcal{Q}} = [(T_\mathcal{P} \cup T_\mathcal{P} \uparrow (\mathcal{A}^I_\mathcal{Q} \setminus \mathcal{A}^I_\mathcal{P})) \cap (T_\mathcal{Q} \cup T_\mathcal{Q} \uparrow (\mathcal{A}^I_\mathcal{P} \setminus \mathcal{A}^I_\mathcal{Q}))] \cap \mathcal{A}^*_{\mathcal{P}\wedge\mathcal{Q}}$
- $F_{\mathcal{P}\wedge\mathcal{Q}} = [(F_\mathcal{P} \cup T_\mathcal{P} \uparrow (\mathcal{A}^I_\mathcal{Q} \setminus \mathcal{A}^I_\mathcal{P})) \cap (F_\mathcal{Q} \cup T_\mathcal{Q} \uparrow (\mathcal{A}^I_\mathcal{P} \setminus \mathcal{A}^I_\mathcal{Q}))] \cap \mathcal{A}^*_{\mathcal{P}\wedge\mathcal{Q}}$.

Intuitively, after any trace of $\mathcal{P} \wedge \mathcal{Q}$, the conjunction must accept any input offered by either $\mathcal{P}$ or $\mathcal{Q}$, but can only issue an output if both $\mathcal{P}$ and $\mathcal{Q}$ are willing to offer it. Once $\mathcal{P}$ becomes inconsistent, or an input is seen that is not an input of $\mathcal{P}$, the conjunction behaves like $\mathcal{Q}$ (and vice-versa).

Quotient. In [4], we introduced a quotient operator acting on components. Given a component $\mathcal{R}$, together with a component $\mathcal{P}$ implementing part of $\mathcal{R}$, the quotient $\mathcal{R}/\mathcal{P}$ yields the coarsest component for the remaining part of $\mathcal{R}$ to be implemented. Thus, the quotient satisfies the property: there exists $\mathcal{Q}$ such that $\mathcal{P} \parallel \mathcal{Q} \sqsubseteq_{imp} \mathcal{R}$ iff $\mathcal{P} \parallel (\mathcal{R}/\mathcal{P}) \sqsubseteq_{imp} \mathcal{R}$ and $\mathcal{Q} \sqsubseteq_{imp} (\mathcal{R}/\mathcal{P})$. Whether the quotient exists depends on the extent to which $\mathcal{P}$ is a sub-component of $\mathcal{R}$.

For the development in this paper, we will not use quotient on components, and refer to [4]. Instead, we will define a quotient operator that acts on AG specifications. Thus, the quotient of two AG specifications yields an AG specification characterising a set of component implementations.

3 Assume-Guarantee Framework for Safety Properties

To support reasoning about components, we introduce the concept of an AG specification, which consists of two prefix-closed sets of traces referred to as the *assumption* and *guarantee*. The assumption specifies the environment's allowable interaction sequences, while the guarantee is a constraint on the component's

behaviour. As assumptions and guarantees are prefix-closed, our theory ensures that components preserve (not necessarily regular) safety properties[3].

Definition 6 (AG specification). *An* AG specification $\mathcal{S}$ *is a tuple* $\langle \mathcal{A}^I_\mathcal{S}, \mathcal{A}^O_\mathcal{S}, \mathcal{R}_\mathcal{S}, \mathcal{G}_\mathcal{S} \rangle$*, in which* $\mathcal{A}^I_\mathcal{S}$ *and* $\mathcal{A}^O_\mathcal{S}$ *are disjoint sets, referred to as the inputs and outputs respectively, and* $\mathcal{R}_\mathcal{S}$ *and* $\mathcal{G}_\mathcal{S}$ *are prefix closed subsets of* $(\mathcal{A}^I_\mathcal{S} \cup \mathcal{A}^O_\mathcal{S})^*$*, referred to as the assumption and guarantee respectively, such that* $t \in \mathcal{R}_\mathcal{S}$ *and* $t' \in (\mathcal{A}^O_\mathcal{S})^*$ *implies* $tt' \in \mathcal{R}_\mathcal{S}$*.*

Since outputs are under the control of a component, we insist that assumptions are closed under output-extensions. On the other hand, we need not insist that the guarantee is closed under input-extensions, since the assumption can select inputs under which the guarantee is given.

Given an AG specification $\mathcal{S}$, we want to be able to say whether a component $\mathcal{P}$ satisfies $\mathcal{S}$. Informally, $\mathcal{P}$ satisfies $\mathcal{S}$ if for any interaction between $\mathcal{P}$ and the environment characterised by a trace t, if $t \in \mathcal{R}_\mathcal{S}$, then $t \in \mathcal{G}_\mathcal{S}$ and t cannot become inconsistent in $\mathcal{P}$ without further stimulation from the environment. Components can thus be thought of as implementations of AG specifications.

Before defining satisfaction, we need to introduce a notion of compatibility between AG specifications and components, meaning that they do not disagree on what are inputs or outputs.

Definition 7 (Compatibility). *Let* $\mathcal{P}$ *be a component, and let* $\mathcal{S}$ *and* $\mathcal{T}$ *be AG-specifications. Then* $\mathcal{P}$ *is compatible with* $\mathcal{S}$*, written* $\mathcal{P} \sim \mathcal{S}$*, iff* $\mathcal{A}^I_\mathcal{P} \cap \mathcal{A}^O_\mathcal{S} = \emptyset = \mathcal{A}^O_\mathcal{P} \cap \mathcal{A}^I_\mathcal{S}$*. Similarly,* $\mathcal{S}$ *is compatible with* $\mathcal{T}$*, written* $\mathcal{S} \sim \mathcal{T}$*, iff* $\mathcal{A}^I_\mathcal{S} \cap \mathcal{A}^O_\mathcal{T} = \emptyset = \mathcal{A}^O_\mathcal{S} \cap \mathcal{A}^I_\mathcal{T}$*.*

We can now give the formal definition for satisfaction of an AG specification by a component.

Definition 8 (AG satisfaction). *A component* $\mathcal{P}$ *satisfies the AG specification* $\mathcal{S}$*, written* $\mathcal{P} \models \mathcal{S}$*, iff:*

S1. $\mathcal{P} \sim \mathcal{S}$
S2. $\mathcal{A}^I_\mathcal{S} \subseteq \mathcal{A}^I_\mathcal{P}$
S3. $\mathcal{A}^O_\mathcal{P} \subseteq \mathcal{A}^O_\mathcal{S}$
S4. $\mathcal{R}_\mathcal{S} \cap T_\mathcal{P} \subseteq \mathcal{G}_\mathcal{S} \cap \overline{F_\mathcal{P}}$*.*

By output-extension closure of assumptions, condition S4 is equivalent to checking $\mathcal{R}_\mathcal{S} \cap T_\mathcal{P} \subseteq \mathcal{G}_\mathcal{S} \cap \overline{F_{\mathcal{E}(\mathcal{P})}}$, which involves the most general safe representation $\mathcal{E}(\mathcal{P})$ of $\mathcal{P}$ (see Definition 2). The following lemma shows that this definition of satisfaction is preserved under the component-based refinement corresponding to safe-substitutivity, subject to compatibility.

Lemma 1. *Let* $\mathcal{P}$ *and* $\mathcal{Q}$ *be components, and let* $\mathcal{S}$ *be an AG specification. If* $\mathcal{P} \models \mathcal{S}$*,* $\mathcal{Q} \sqsubseteq_{imp} \mathcal{P}$ *and* $\mathcal{Q} \sim \mathcal{S}$*, then* $\mathcal{Q} \models \mathcal{S}$*.*

[3] Model-checking components against AG specifications would force us to restrict the properties we can encode and check. In this setting, we would naturally restrict to the regular safety properties, which can be encoded by finite-state automata.

3.1 Refinement

There is a natural hierarchy on AG specifications respecting the satisfaction rule defined in Definition 8. From this we can define a refinement relation on AG specifications that corresponds to implementation containment. But first, we introduce the shorthand: $\mathsf{violations}(X) \triangleq \{t \in \mathcal{A}_X^* : \exists t' \in (\mathcal{A}_X^I)^* \cdot tt' \in \mathcal{R}_X \cap \overline{\mathcal{G}_X}\}\mathcal{A}_X^*$.

Definition 9 (AG refinement). *Let $\mathcal{S}$ and $\mathcal{T}$ be AG specifications. $\mathcal{S}$ is said to be a refinement of $\mathcal{T}$, written $\mathcal{S} \sqsubseteq \mathcal{T}$, iff:*

R1. $\mathcal{S} \sim \mathcal{T}$
R2. $\mathcal{A}_\mathcal{T}^I \subseteq \mathcal{A}_\mathcal{S}^I$
R3. $\mathcal{A}_\mathcal{S}^O \subseteq \mathcal{A}_\mathcal{T}^O$
R4. $\mathsf{violations}(\mathcal{T}) \cap \mathcal{A}_\mathcal{S}^* \subseteq \mathsf{violations}(\mathcal{S})$
R5. $\mathcal{R}_\mathcal{T} \cap \mathcal{A}_\mathcal{S}^* \subseteq \mathcal{R}_\mathcal{S} \cup \mathsf{violations}(\mathcal{S})$.

It is our intention that $\mathcal{S} \sqsubseteq \mathcal{T}$ iff the implementations of $\mathcal{S}$ are contained within the implementations of $\mathcal{T}$ (subject to compatibility). Conditions R1-R3 are the bare minimum to uphold this principle. For condition R4, any component having a trace $t \in \mathsf{violations}(\mathcal{T}) \cap \mathcal{A}_\mathcal{S}^*$ cannot be an implementation of $\mathcal{T}$, so it should not be an implementation of $\mathcal{S}$. For this to be the case, the component must violate the guarantee on $\mathcal{S}$, i.e., $t \in \mathsf{violations}(\mathcal{S})$. Condition R5 deals with inconsistent traces. If a component has an inconsistent trace $t \in \mathcal{R}_\mathcal{T} \cap \mathcal{A}_\mathcal{S}^*$, then this cannot be an implementation of $\mathcal{T}$. Consequently, the component must not be an implementation of $\mathcal{S}$, so either t must violate the guarantee of $\mathcal{S}$, i.e., $t \in \mathsf{violations}(\mathcal{S})$, or t must be in $\mathcal{R}_\mathcal{S}$, so that the component cannot satisfy $\mathcal{S}$.

Lemma 2. *Refinement respects implementation containment:*

$$\mathcal{S} \sqsubseteq \mathcal{T} \iff \{\mathcal{P} : \mathcal{P} \models \mathcal{S} \text{ and } \mathcal{P} \sim \mathcal{T}\} \subseteq \{\mathcal{P} : \mathcal{P} \models \mathcal{T}\}.$$

In [15], Larsen *et al.* give a sound and complete characterisation of their refinement relation (which corresponds to implementation containment, as for us) by means of conformance tests. The definition assumes equality of interfaces, so does not need to deal with issues of compatibility or the complexities of both covariant and contravariant inclusion of inputs and outputs respectively (i.e., conditions R1-R3). Thus, their definition largely corresponds to condition R4. Condition R5 is not necessary in that setting, as implementation models are required to be input-enabled.

Refinement can be shown to be a preorder, provided that we add the minor technical condition that compatibility of components is maintained, as the next lemma shows.

Lemma 3 (Weak transitivity). *For AG specifications $\mathcal{S}$, $\mathcal{T}$ and $\mathcal{U}$, if $\mathcal{S} \sqsubseteq \mathcal{T}$, $\mathcal{T} \sqsubseteq \mathcal{U}$ and $\mathcal{S} \sim \mathcal{U}$, then $\mathcal{S} \sqsubseteq \mathcal{U}$.*

As an aside, component-based refinement $\sqsubseteq_{imp}$ is a preorder because, in refining a component $\mathcal{P}$ to a component $\mathcal{Q}$, it is possible to transform some of $\mathcal{P}$'s outputs into inputs of $\mathcal{Q}$, as this preserves safe-substitutivity. However, this transformation of action types does not make sense with AG specifications, which talk explicitly about the behaviour of the environment.

3.2 Inferring Components from AG Specifications

Given a specification for a component, we require a way for developers to construct an actual component that satisfies the requirements of the specification. In the following definition, we show how to infer the least refined component that satisfies a given specification.

Definition 10 (Inferred component). *Let $\mathcal{S}$ be an AG specification. Then the least refined implementation of $\mathcal{S}$ is the component $\mathcal{I}(\mathcal{S}) = \langle \mathcal{A}^I_{\mathcal{S}}, \mathcal{A}^O_{\mathcal{S}}, T_{\mathcal{I}(\mathcal{S})}, F_{\mathcal{I}(\mathcal{S})} \rangle$, defined only when $\epsilon \in T_{\mathcal{I}(\mathcal{S})}$, where:*

- $T_{\mathcal{I}(\mathcal{S})} = \mathsf{pre}(\{t \in \mathcal{R}_{\mathcal{S}} \cap \mathcal{G}_{\mathcal{S}} : \forall t' \in (\mathcal{A}^I_{\mathcal{S}})^* \cdot tt' \in \overline{\mathcal{R}_{\mathcal{S}}} \cup \mathcal{G}_{\mathcal{S}}\}) \cup F_{\mathcal{I}(\mathcal{S})}$
- $F_{\mathcal{I}(\mathcal{S})} = \{tit' : t \in \mathcal{R}_{\mathcal{S}} \cap \mathcal{G}_{\mathcal{S}},\ i \in \mathcal{A}^I_{\mathcal{S}} \text{ and } ti \notin \mathcal{R}_{\mathcal{S}}\} \cup \{t \in \mathcal{A}^*_{\mathcal{S}} : \epsilon \notin \mathcal{R}_{\mathcal{S}}\}$.

The following lemma shows that the obtained component model really is least refined with respect to the refinement preorder $\sqsubseteq_{imp}$ on implementations.

Lemma 4. *Let $\mathcal{S}$ be an AG specification, and let $\mathcal{P}$ be a component. Then:*

- *$\epsilon \notin T_{\mathcal{I}(\mathcal{S})}$ implies $\mathcal{S}$ is non-implementable;*
- *$\epsilon \in T_{\mathcal{I}(\mathcal{S})}$ implies $\mathcal{I}(\mathcal{S}) \models \mathcal{S}$; and*
- *$\mathcal{P} \models \mathcal{S}$ iff $\mathcal{P} \sqsubseteq_{imp} \mathcal{I}(\mathcal{S})$.*

3.3 Characteristic AG Specification of a Component

One may be interested in the most general AG specification that satisfies a component, which we refer to as the characteristic AG specification of the component. This can be found by examining the component's safe traces.

Definition 11 (Characteristic AG specification). *The* characteristic AG specification *for the component $\mathcal{P}$ is an AG specification $\mathcal{AG}(\mathcal{P}) = \langle \mathcal{A}^I_{\mathcal{P}}, \mathcal{A}^O_{\mathcal{P}}, \mathcal{R}_{\mathcal{AG}(\mathcal{P})}, \mathcal{G}_{\mathcal{AG}(\mathcal{P})} \rangle$, where $\mathcal{R}_{\mathcal{AG}(\mathcal{P})} = \mathcal{A}^*_{\mathcal{P}} \setminus F_{\mathcal{E}(\mathcal{P})}$ and $\mathcal{G}_{\mathcal{AG}(\mathcal{P})} = T_{\mathcal{P}} \setminus F_{\mathcal{E}(\mathcal{P})}$.*

The largest assumption safe for component $\mathcal{P}$ is the set of all non-inconsistent traces, while the guarantee is the set of traces of $\mathcal{E}(\mathcal{P})$ that are non-inconsistent. As the following lemma demonstrates, the characteristic AG specification satisfies the desired properties.

Lemma 5. *Let $\mathcal{P}$ be a component and let $\mathcal{S}$ be an AG specification. Then:*

- *$\mathcal{P} \models \mathcal{AG}(\mathcal{P})$; and*
- *$\mathcal{P} \models \mathcal{S}$ iff $\mathcal{AG}(\mathcal{P}) \sqsubseteq \mathcal{S}$.*

The final point in the previous lemma shows that satisfaction of a specification by a component is equivalent to checking whether the characteristic AG specification of the component is a refinement of the specification. This means that implementability of specifications built up compositionally follows immediately from compositionality results on AG specifications, as we will see in the subsequent sections.

We are now in a position to present sound and complete AG rules for inferring properties of composite systems from the properties of their sub-components.

3.4 Parallel Composition

The AG rule for parallel composition is based on the well-established theorem of Abadi and Lamport [7], which has appeared in several forms [19,20,21]. Intuitively, the guarantee of any component must not be allowed to violate the assumptions of the other components. Such reasoning seems circular, but the circularity can be broken up in our setting as a safety property cannot be simultaneously violated by two or more components. This is due to an output being under the control of at most one component.

Notation. To assist in our definition, we introduce the following shorthands:

- $\mathcal{R}(\mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{Q}) \triangleq (\mathcal{R}_{\mathcal{S}_\mathcal{P}} \Uparrow \mathcal{A}_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}}) \cap (\mathcal{R}_{\mathcal{S}_\mathcal{Q}} \Uparrow \mathcal{A}_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}})$
- $\mathcal{G}(\mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{Q}) \triangleq (\mathcal{G}_{\mathcal{S}_\mathcal{P}} \Uparrow \mathcal{A}_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}}) \cap (\mathcal{G}_{\mathcal{S}_\mathcal{Q}} \Uparrow \mathcal{A}_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}})$
- $\mathcal{G}^+(\mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{Q}) \triangleq (\mathcal{G}_{\mathcal{S}_\mathcal{P}} \Uparrow\Uparrow \mathcal{A}_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}}) \cap (\mathcal{G}_{\mathcal{S}_\mathcal{Q}} \Uparrow\Uparrow \mathcal{A}_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}})$.

Definition 12. *Let $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{Q}$ be AG specifications such that $\mathcal{A}^O_{\mathcal{S}_\mathcal{P}} \cap \mathcal{A}^O_{\mathcal{S}_\mathcal{Q}} = \emptyset$. If $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{Q}$ are both implementable, then $\mathcal{S}_\mathcal{P} \,||\, \mathcal{S}_\mathcal{Q}$ is an AG specification $\langle \mathcal{A}^I_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}}, \mathcal{A}^O_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}}, \mathcal{R}_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}}, \mathcal{G}_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}} \rangle$ defined by:*

- $\mathcal{A}^I_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}} = (\mathcal{A}^I_{\mathcal{S}_\mathcal{P}} \cup \mathcal{A}^I_{\mathcal{S}_\mathcal{Q}}) \setminus (\mathcal{A}^O_{\mathcal{S}_\mathcal{P}} \cup \mathcal{A}^O_{\mathcal{S}_\mathcal{Q}})$
- $\mathcal{A}^O_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}} = \mathcal{A}^O_{\mathcal{S}_\mathcal{P}} \cup \mathcal{A}^O_{\mathcal{S}_\mathcal{Q}}$
- $\mathcal{R}_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}} \subseteq \mathcal{A}^*_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}}$ *is the largest prefix closed set satisfying* $\mathcal{R}_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}}(\mathcal{A}^O_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}})^* \cap \mathcal{G}^+(\mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{Q}) \subseteq \mathcal{R}(\mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{Q})$
- $\mathcal{G}_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}} = \mathcal{R}_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}} \cap \mathcal{G}(\mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{Q})$.

If at least one of $\mathcal{S}_\mathcal{P}$ or $\mathcal{S}_\mathcal{Q}$ is non-implementable, then $\mathcal{S}_\mathcal{P} \,||\, \mathcal{S}_\mathcal{Q} = \langle \mathcal{A}^I_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}}, \mathcal{A}^O_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}}, \mathcal{A}^*_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}}, \emptyset \rangle$

$\mathcal{S}_\mathcal{P} \,||\, \mathcal{S}_\mathcal{Q}$ yields the strongest specification satisfiable by the parallel composition of any two components that satisfy $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{Q}$. The specification only guarantees what can be assured by both $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{Q}$, thus it is the strongest composition. The assumption is the largest collection of environmental behaviours that cannot violate either of the guarantees $\mathcal{G}_{\mathcal{S}_\mathcal{P}}$ or $\mathcal{G}_{\mathcal{S}_\mathcal{Q}}$, and moreover does not permit a component implementing one of the specifications to violate the other specification's assumption. Ignoring differences in alphabets, this can loosely be phrased as $\mathcal{R}_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}} \cap \mathcal{G}_{\mathcal{S}_\mathcal{P}} \subseteq \mathcal{R}_{\mathcal{S}_\mathcal{Q}}$ and $\mathcal{R}_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}} \cap \mathcal{G}_{\mathcal{S}_\mathcal{Q}} \subseteq \mathcal{R}_{\mathcal{S}_\mathcal{P}}$, which is akin to the presentation in [7]. However, as implementations are not required to be input-enabled, this must be reformulated as $\mathcal{R}_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}} \cap \mathcal{G}^+(\mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{Q}) \subseteq \mathcal{R}(\mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{Q})$.

The set $\mathcal{G}^+(\mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{Q})$ extends $\mathcal{G}(\mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{Q})$ by a single input on each of $\mathcal{G}_{\mathcal{S}_\mathcal{P}}$ and $\mathcal{G}_{\mathcal{S}_\mathcal{Q}}$, and also includes ϵ. This has the effect of ensuring that, if $t \in \mathcal{G}^+(\mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{Q}) \cap \mathcal{R}(\mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{Q})$ and $ta \notin \mathcal{G}^+(\mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{Q})$, then whatever the action type of a, wlog $t \upharpoonright \mathcal{A}_{\mathcal{S}_\mathcal{P}} \in \mathcal{R}_{\mathcal{S}_\mathcal{P}} \cap \overline{\mathcal{G}_{\mathcal{S}_\mathcal{P}}}$ or $ta \upharpoonright \mathcal{A}_{\mathcal{S}_\mathcal{P}} \in \mathcal{R}_{\mathcal{S}_\mathcal{P}} \cap \overline{\mathcal{G}_{\mathcal{S}_\mathcal{P}}}$. Thus, any implementation of $\mathcal{S}_\mathcal{P}$ must have suppressed an output at some stage along the trace $ta \upharpoonright \mathcal{A}_{\mathcal{S}_\mathcal{P}}$, implying the parallel composition of any two implementations of $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{Q}$ will suppress an output along ta. Thus, $\mathcal{R}_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}}$ contains only traces within $\mathcal{G}_{\mathcal{S}_\mathcal{P}||\mathcal{S}_\mathcal{Q}}$ and traces not reachable by any pair of implementations of $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{Q}$.

Subject to suitable constraints on the alphabets of AG specifications, it can be shown that the parallel composition operator on AG specifications is compositional under the AG refinement relation, as the following theorem demonstrates.

Theorem 1. *Let $\mathcal{S}_\mathcal{P}$, $\mathcal{S}'_\mathcal{P}$, $\mathcal{S}_\mathcal{Q}$ and $\mathcal{S}'_\mathcal{Q}$ be AG specifications such that $\mathcal{A}^O_{\mathcal{S}_\mathcal{P}} \cap \mathcal{A}^O_{\mathcal{S}_\mathcal{Q}} = \emptyset$, $\mathcal{S}'_\mathcal{P} \parallel \mathcal{S}'_\mathcal{Q} \sim \mathcal{S}_\mathcal{P} \parallel \mathcal{S}_\mathcal{Q}$, $\mathcal{A}^I_{\mathcal{S}'_\mathcal{P}} \cap \mathcal{A}^O_{\mathcal{S}'_\mathcal{Q}} \subseteq \mathcal{A}^I_{\mathcal{S}_\mathcal{P}} \cap \mathcal{A}^O_{\mathcal{S}_\mathcal{Q}}$, $\mathcal{A}^O_{\mathcal{S}'_\mathcal{P}} \cap \mathcal{A}^I_{\mathcal{S}'_\mathcal{Q}} \subseteq \mathcal{A}^O_{\mathcal{S}_\mathcal{P}} \cap \mathcal{A}^I_{\mathcal{S}_\mathcal{Q}}$ and $\mathcal{A}^I_{\mathcal{S}'_\mathcal{P}} \cap \mathcal{A}^I_{\mathcal{S}'_\mathcal{Q}} \cap \mathcal{A}^I_{\mathcal{S}_\mathcal{P} \parallel \mathcal{S}_\mathcal{Q}} \subseteq \mathcal{A}^I_{\mathcal{S}_\mathcal{P}} \cap \mathcal{A}^I_{\mathcal{S}_\mathcal{Q}}$. If $\mathcal{S}'_\mathcal{P} \sqsubseteq \mathcal{S}_\mathcal{P}$ and $\mathcal{S}'_\mathcal{Q} \sqsubseteq \mathcal{S}_\mathcal{Q}$, then $\mathcal{S}'_\mathcal{P} \parallel \mathcal{S}'_\mathcal{Q} \sqsubseteq \mathcal{S}_\mathcal{P} \parallel \mathcal{S}_\mathcal{Q}$.*

The condition $\mathcal{A}^O_{\mathcal{S}_\mathcal{P}} \cap \mathcal{A}^O_{\mathcal{S}_\mathcal{Q}} = \emptyset$ ensures that the parallel composition of the AG specifications is defined, while $\mathcal{S}'_\mathcal{P} \parallel \mathcal{S}'_\mathcal{Q} \sim \mathcal{S}_\mathcal{P} \parallel \mathcal{S}_\mathcal{Q}$ means $\mathcal{S}'_\mathcal{P} \parallel \mathcal{S}'_\mathcal{Q}$ and $\mathcal{S}_\mathcal{P} \parallel \mathcal{S}_\mathcal{Q}$ are comparable under refinement. The remaining three conditions are standard for compositionality of parallel composition. From this compositionality result, it is easy to give a sound and complete AG rule.

Theorem 2. *Let $\mathcal{P}$ and $\mathcal{Q}$ be components, and let $\mathcal{S}_\mathcal{P}$, $\mathcal{S}_\mathcal{Q}$ and $\mathcal{S}$ be AG specifications such that $\mathcal{P} \parallel \mathcal{Q} \sim \mathcal{S}$, $\mathcal{A}^I_\mathcal{P} \cap \mathcal{A}^O_\mathcal{Q} \subseteq \mathcal{A}^I_{\mathcal{S}_\mathcal{P}} \cap \mathcal{A}^O_{\mathcal{S}_\mathcal{Q}}$, $\mathcal{A}^O_\mathcal{P} \cap \mathcal{A}^I_\mathcal{Q} \subseteq \mathcal{A}^O_{\mathcal{S}_\mathcal{P}} \cap \mathcal{A}^I_{\mathcal{S}_\mathcal{Q}}$ and $\mathcal{A}^I_\mathcal{P} \cap \mathcal{A}^I_\mathcal{Q} \cap \mathcal{A}^I_{\mathcal{S}_\mathcal{P} \parallel \mathcal{S}_\mathcal{Q}} \subseteq \mathcal{A}^I_{\mathcal{S}_\mathcal{P}} \cap \mathcal{A}^I_{\mathcal{S}_\mathcal{Q}}$. Then the following AG rule is both sound and complete:*

$$\textsc{Parallel}\ \frac{\mathcal{P} \models \mathcal{S}_\mathcal{P} \qquad \mathcal{Q} \models \mathcal{S}_\mathcal{Q} \qquad \mathcal{S}_\mathcal{P} \parallel \mathcal{S}_\mathcal{Q} \sqsubseteq \mathcal{S}}{\mathcal{P} \parallel \mathcal{Q} \models \mathcal{S}}.$$

3.5 Conjunction

In this section we define a conjunctive operator on AG specifications for combining independently developed requirements. From this we show that the operator is both compositional and corresponds to the meet operation on the refinement relation. This allows us to formulate a sound and complete AG rule.

The conjunction of AG specifications $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{Q}$ is only defined when $\mathcal{A}^I_{\mathcal{S}_\mathcal{P}} \cup \mathcal{A}^I_{\mathcal{S}_\mathcal{Q}}$ is disjoint from $\mathcal{A}^O_{\mathcal{S}_\mathcal{P}} \cup \mathcal{A}^O_{\mathcal{S}_\mathcal{Q}}$, in which case we say $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{Q}$ are composable. The composability constraint is necessary, as otherwise it is not possible to find an interface that can refine both $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{Q}$.

Definition 13. *Let $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{Q}$ be AG specifications composable for conjunction. Then $\mathcal{S}_\mathcal{P} \wedge \mathcal{S}_\mathcal{Q}$ is an AG specification $\langle \mathcal{A}^I_{\mathcal{S}_\mathcal{P} \wedge \mathcal{S}_\mathcal{Q}}, \mathcal{A}^O_{\mathcal{S}_\mathcal{P} \wedge \mathcal{S}_\mathcal{Q}}, \mathcal{R}_{\mathcal{S}_\mathcal{P} \wedge \mathcal{S}_\mathcal{Q}}, \mathcal{G}_{\mathcal{S}_\mathcal{P} \wedge \mathcal{S}_\mathcal{Q}} \rangle$ defined by:*

- $\mathcal{A}^I_{\mathcal{S}_\mathcal{P} \wedge \mathcal{S}_\mathcal{Q}} = \mathcal{A}^I_{\mathcal{S}_\mathcal{P}} \cup \mathcal{A}^I_{\mathcal{S}_\mathcal{Q}}$
- $\mathcal{A}^O_{\mathcal{S}_\mathcal{P} \wedge \mathcal{S}_\mathcal{Q}} = \mathcal{A}^O_{\mathcal{S}_\mathcal{P}} \cap \mathcal{A}^O_{\mathcal{S}_\mathcal{Q}}$
- $\mathcal{R}_{\mathcal{S}_\mathcal{P} \wedge \mathcal{S}_\mathcal{Q}} = (\mathcal{R}_{\mathcal{S}_\mathcal{P}} \cup \mathcal{R}_{\mathcal{S}_\mathcal{Q}}) \cap \mathcal{A}^*_{\mathcal{S}_\mathcal{P} \wedge \mathcal{S}_\mathcal{Q}}$
- $\mathcal{G}_{\mathcal{S}_\mathcal{P} \wedge \mathcal{S}_\mathcal{Q}}$ *is the intersection of the following sets:*
  - $\mathcal{R}_{\mathcal{S}_\mathcal{P} \wedge \mathcal{S}_\mathcal{Q}} \cap (\mathcal{G}_{\mathcal{S}_\mathcal{P}} \cup \mathcal{G}_{\mathcal{S}_\mathcal{Q}})$
  - $\mathsf{pre}(\overline{\mathcal{R}_{\mathcal{S}_\mathcal{P}}} \cup \mathcal{G}_{\mathcal{S}_\mathcal{P}}) \uparrow_\epsilon (\mathcal{A}^I_{\mathcal{S}_\mathcal{Q}} \setminus \mathcal{A}^I_{\mathcal{S}_\mathcal{P}})$
  - $\mathsf{pre}(\overline{\mathcal{R}_{\mathcal{S}_\mathcal{Q}}} \cup \mathcal{G}_{\mathcal{S}_\mathcal{Q}}) \uparrow_\epsilon (\mathcal{A}^I_{\mathcal{S}_\mathcal{P}} \setminus \mathcal{A}^I_{\mathcal{S}_\mathcal{Q}})$.

The assumption $\mathcal{R}_{\mathcal{S}_\mathcal{P} \wedge \mathcal{S}_\mathcal{Q}}$ is constrained to be within at least one of $\mathcal{R}_{\mathcal{S}_\mathcal{P}}$ or $\mathcal{R}_{\mathcal{S}_\mathcal{Q}}$. On the other hand, the guarantee $\mathcal{G}_{\mathcal{S}_\mathcal{P} \wedge \mathcal{S}_\mathcal{Q}}$ must be within at least one of $\mathcal{G}_{\mathcal{S}_\mathcal{P}}$ or $\mathcal{G}_{\mathcal{S}_\mathcal{Q}}$, and must ensure that, if the assumption for one of the specifications is satisfied, then the corresponding guarantee cannot have been violated.

The next two theorems show that our definition of conjunction corresponds to the meet operator on the refinement relation, and is compositional under refinement. Consequently, the set of implementations for $\mathcal{S}_\mathcal{P} \wedge \mathcal{S}_\mathcal{Q}$ is the intersection of the implementation sets for $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{Q}$.

Theorem 3. *Let $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{Q}$ be AG specifications such that $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{Q}$ are composable for conjunction. Then:*

- $\mathcal{S}_\mathcal{P} \wedge \mathcal{S}_\mathcal{Q} \sqsubseteq \mathcal{S}_\mathcal{P}$
- $\mathcal{S}_\mathcal{P} \wedge \mathcal{S}_\mathcal{Q} \sqsubseteq \mathcal{S}_\mathcal{Q}$
- $\mathcal{S}_\mathcal{R} \sqsubseteq \mathcal{S}_\mathcal{P}$ *and* $\mathcal{S}_\mathcal{R} \sqsubseteq \mathcal{S}_\mathcal{Q}$ *implies* $\mathcal{S}_\mathcal{R} \sqsubseteq \mathcal{S}_\mathcal{P} \wedge \mathcal{S}_\mathcal{Q}$.

Theorem 4. *Let $\mathcal{S}_\mathcal{P}$, $\mathcal{S}_\mathcal{Q}$, $\mathcal{S}'_\mathcal{P}$ and $\mathcal{S}'_\mathcal{Q}$ be AG specifications such that $\mathcal{S}'_\mathcal{P}$ and $\mathcal{S}'_\mathcal{Q}$ are composable for conjunction, $\mathcal{S}'_\mathcal{P} \sim \mathcal{S}_\mathcal{Q}$ and $\mathcal{S}'_\mathcal{Q} \sim \mathcal{S}_\mathcal{P}$. If $\mathcal{S}'_\mathcal{P} \sqsubseteq \mathcal{S}_\mathcal{P}$ and $\mathcal{S}'_\mathcal{Q} \sqsubseteq \mathcal{S}_\mathcal{Q}$, then $\mathcal{S}'_\mathcal{P} \wedge \mathcal{S}'_\mathcal{Q} \sqsubseteq \mathcal{S}_\mathcal{P} \wedge \mathcal{S}_\mathcal{Q}$.*

From these strong algebraic properties, we can formulate an AG rule for conjunction that is both sound and complete.

Theorem 5. *Let $\mathcal{P}$ and $\mathcal{Q}$ be components composable for conjunction, and let $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{Q}$ be AG specifications such that $\mathcal{P} \sim \mathcal{S}_\mathcal{Q}$, $\mathcal{Q} \sim \mathcal{S}_\mathcal{P}$ and $\mathcal{P} \wedge \mathcal{Q} \sim \mathcal{S}$. Then the following AG rule is both sound and complete:*

$$\textsc{Conjunction}\ \frac{\mathcal{P} \models \mathcal{S}_\mathcal{P} \qquad \mathcal{Q} \models \mathcal{S}_\mathcal{Q} \qquad \mathcal{S}_\mathcal{P} \wedge \mathcal{S}_\mathcal{Q} \sqsubseteq \mathcal{S}}{\mathcal{P} \wedge \mathcal{Q} \models \mathcal{S}}.$$

3.6 Quotient

The AG rule for parallel composition in Theorem 2 makes use of the composition $\mathcal{S}_\mathcal{P} \,||\, \mathcal{S}_\mathcal{Q}$. To support incremental development, we need a way of decomposing the composition to find $\mathcal{S}_\mathcal{Q}$ given $\mathcal{S}_\mathcal{P}$. We can do this using a quotient operator.

Definition 14. *Let $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{W}$ be AG specifications. Then the quotient $\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}$ is an AG specification $\langle \mathcal{A}^I_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}}, \mathcal{A}^O_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}}, \mathcal{R}_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}}, \mathcal{G}_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}} \rangle$, defined only when $\mathcal{A}^O_{\mathcal{S}_\mathcal{P}} \subseteq \mathcal{A}^O_{\mathcal{S}_\mathcal{W}}$, where $\mathcal{A}^I_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}} = \mathcal{A}^I_{\mathcal{S}_\mathcal{W}} \setminus \mathcal{A}^I_{\mathcal{S}_\mathcal{P}}$, $\mathcal{A}^O_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}} = \mathcal{A}^O_{\mathcal{S}_\mathcal{W}} \setminus \mathcal{A}^O_{\mathcal{S}_\mathcal{P}}$ and:*

- *If $\mathcal{S}_\mathcal{P}$ is implementable, and $\epsilon \in \mathcal{R}_{\mathcal{S}_\mathcal{W}}$ implies $\epsilon \in \mathcal{R}_{\mathcal{S}_\mathcal{P}}$, then:*
  - $\mathcal{R}_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}} = [\mathcal{R}_{\mathcal{S}_\mathcal{W}} \cap (\mathcal{G}_{\mathcal{S}_\mathcal{P}} \uparrow\uparrow \mathcal{A}_{\mathcal{S}_\mathcal{W}})(\mathcal{A}^O_{\mathcal{S}_\mathcal{W}})^*] \restriction \mathcal{A}_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}}$
  - $\mathcal{G}_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}} = \mathcal{R}_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}} \cap (X \restriction \mathcal{A}_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}})$, *where X is the largest prefix closed set satisfying* $X(\mathcal{A}^I_{\mathcal{S}_\mathcal{P}})^* \cap \mathcal{R}_{\mathcal{S}_\mathcal{W}} \subseteq \mathsf{pre}(\mathcal{G}_{\mathcal{S}_\mathcal{W}} \cup \overline{\mathcal{G}_{\mathcal{S}_\mathcal{P}} \Uparrow \mathcal{A}_{\mathcal{S}_\mathcal{W}}}) \cap \mathsf{pre}((\mathcal{R}_{\mathcal{S}_\mathcal{P}} \Uparrow \mathcal{A}_{\mathcal{S}_\mathcal{W}}) \cup \overline{\mathcal{G}_{\mathcal{S}_\mathcal{P}} \uparrow\uparrow \mathcal{A}_{\mathcal{S}_\mathcal{W}}})$.
- *If $\mathcal{S}_\mathcal{P}$ is implementable and $\epsilon \in \mathcal{R}_{\mathcal{S}_\mathcal{W}} \cap \overline{\mathcal{R}_{\mathcal{S}_\mathcal{P}}}$, then $\mathcal{R}_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}} = \mathcal{A}^*_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}}$ and $\mathcal{G}_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}} = \emptyset$.*
- *If $\mathcal{S}_\mathcal{P}$ is non-implementable, then $\mathcal{R}_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}} = \mathcal{G}_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}} = \emptyset$.*

Although not immediately obvious, the assumption in the previous definition is closed under output-extensions. Before explaining the definition, we introduce the following theorem, which shows that the quotient operator on AG specifications yields the weakest decomposition of the parallel composition.

Theorem 6. *Let $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{W}$ be AG specifications. Then there exists an AG specification $\mathcal{S}_\mathcal{Q}$ such that $\mathcal{S}_\mathcal{P} \parallel \mathcal{S}_\mathcal{Q} \sqsubseteq \mathcal{S}_\mathcal{W}$ iff the following properties hold:*

- *The quotient $\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}$ is defined*
- *$\mathcal{S}_\mathcal{P} \parallel (\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}) \sqsubseteq \mathcal{S}_\mathcal{W}$*
- *$\mathcal{S}_\mathcal{Q} \sqsubseteq \mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}$.*

To make sense of the definition for quotient (in the difficult case of $\mathcal{S}_\mathcal{P}$ being implementable and $\epsilon \in \mathcal{R}_{\mathcal{S}_\mathcal{W}}$ implies $\epsilon \in \mathcal{R}_{\mathcal{S}_\mathcal{P}}$), it is necessary to consider the final two results in Theorem 6. For these, we need to show that: (i) $\mathcal{R}_{\mathcal{S}_\mathcal{W}} \subseteq \mathcal{R}_{\mathcal{S}_\mathcal{P} \parallel (\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P})}$; and (ii) $\mathcal{R}_{\mathcal{S}_\mathcal{W}} \cap \overline{\mathcal{G}_{\mathcal{S}_\mathcal{W}}} \subseteq \mathsf{violations}(\mathcal{S}_\mathcal{P} \parallel (\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}))$. Clause (i) amounts to showing $\mathcal{R}_{\mathcal{S}_\mathcal{W}} \cap \mathcal{G}^+(\mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}) \subseteq \mathcal{R}(\mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P})$, i.e., the condition for parallel composition. Thus, the assumption $\mathcal{R}_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}}$ is the smallest output-closed set such that $t \in \mathcal{R}_{\mathcal{S}_\mathcal{W}}$ and $t \in \mathcal{G}_{\mathcal{S}_\mathcal{P}} \upuparrows \mathcal{A}_{\mathcal{S}_\mathcal{W}}$ implies $t \in \mathcal{R}_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}} \Uparrow \mathcal{A}_{\mathcal{S}_\mathcal{W}}$. The cases of $t \notin \mathcal{R}_{\mathcal{S}_\mathcal{P}} \Uparrow \mathcal{A}_{\mathcal{S}_\mathcal{W}}$ or $t \notin \mathcal{G}_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}} \upuparrows \mathcal{A}_{\mathcal{S}_\mathcal{W}}$ are handled by $\mathcal{G}_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}}$.

Considering the guarantee $\mathcal{G}_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}}$, it is obvious that it need only be contained within the assumption $\mathcal{R}_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}}$. Moreover, it is safe to have $t \in \mathcal{G}_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}} \upuparrows \mathcal{A}_{\mathcal{S}_\mathcal{W}}$ if $t \notin \mathcal{G}_{\mathcal{S}_\mathcal{P}} \upuparrows \mathcal{A}_{\mathcal{S}_\mathcal{W}}$ or $t \in \mathcal{R}_{\mathcal{S}_\mathcal{P}} \Uparrow_{\mathcal{A}_{\mathcal{S}_\mathcal{P}}} \mathcal{A}_{\mathcal{S}_\mathcal{W}}$; this is equivalent to requiring $t \in \mathsf{pre}((\mathcal{R}_{\mathcal{S}_\mathcal{P}} \Uparrow \mathcal{A}_{\mathcal{S}_\mathcal{W}}) \cup \overline{\mathcal{G}_{\mathcal{S}_\mathcal{P}} \upuparrows \mathcal{A}_{\mathcal{S}_\mathcal{W}}})$. For requirement (ii), if $t \in \mathcal{G}_{\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}} \Uparrow \mathcal{A}_{\mathcal{S}_\mathcal{W}}$, then it must be the case that $t \notin \mathcal{G}_{\mathcal{S}_\mathcal{W}}$ implies $t \notin \mathcal{G}_{\mathcal{S}_\mathcal{P}} \Uparrow \mathcal{A}_{\mathcal{S}_\mathcal{W}}$. This is equivalent to requiring $t \in \mathsf{pre}(\mathcal{G}_{\mathcal{S}_\mathcal{W}} \cup \overline{\mathcal{G}_{\mathcal{S}_\mathcal{P}} \Uparrow \mathcal{A}_{\mathcal{S}_\mathcal{W}}})$. Piecing these conditions together yields a definition of quotient that is correct by construction.

Theorem 7. *Let $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{W}$ be AG specifications such that $\mathcal{P}$ ranges over components having the same interface as $\mathcal{S}_\mathcal{P}$, and $\mathcal{Q}$ is a component having the same interface as $\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}$. If $\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}$ is defined (i.e., $\mathcal{A}^Q_{\mathcal{S}_\mathcal{P}} \subseteq \mathcal{A}^Q_{\mathcal{S}_\mathcal{W}}$), then the following AG rule is sound and complete:*

$$\textsc{Quotient}\ \frac{\forall \mathcal{P} \cdot \mathcal{P} \models \mathcal{S}_\mathcal{P} \text{ implies } \mathcal{P} \parallel \mathcal{Q} \models \mathcal{S}_\mathcal{W}}{\mathcal{Q} \models \mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}}.$$

The restriction on $\mathcal{P}$ and $\mathcal{S}_\mathcal{P}$ having the same interface, and $\mathcal{Q}$ and $\mathcal{S}_\mathcal{W}/\mathcal{S}_\mathcal{P}$ having the same interface, is necessary, because the parallel operator is only compositional under certain restrictions on the interfaces (cf Theorem 1).

3.7 Decomposing Parallel Composition

The following corollary shows how we can revise the AG rule for parallel composition so that it makes use of quotient on AG specifications when we know the global specification $\mathcal{S}$. This is useful for system development, as we will often have the specification of a global system, rather than the specifications of the systems to be composed.

Corollary 1. *Let $\mathcal{P}$ and $\mathcal{Q}$ be components such that $\mathcal{A}^I_\mathcal{P} \cap \mathcal{A}^I_\mathcal{Q} = \emptyset$, and let $\mathcal{S}_\mathcal{P}$, $\mathcal{S}_\mathcal{Q}$ and $\mathcal{S}$ be AG specifications. If $\mathcal{A}^O_{\mathcal{S}_\mathcal{P}} \cap \mathcal{A}^O_{\mathcal{S}_\mathcal{Q}} = \emptyset$, $\mathcal{P} \parallel \mathcal{Q} \sim \mathcal{S}$, $\mathcal{A}^I_\mathcal{P} \cap \mathcal{A}^O_\mathcal{Q} \subseteq \mathcal{A}^I_{\mathcal{S}_\mathcal{P}} \cap \mathcal{A}^O_{\mathcal{S}_\mathcal{Q}}$ and $\mathcal{A}^O_\mathcal{P} \cap \mathcal{A}^I_\mathcal{Q} \subseteq \mathcal{A}^O_{\mathcal{S}_\mathcal{P}} \cap \mathcal{A}^I_{\mathcal{S}_\mathcal{Q}}$, then the following rule is both sound and complete:*

$$\textsc{Parallel-Decompose}\ \frac{\mathcal{P} \models \mathcal{S}_\mathcal{P} \qquad \mathcal{Q} \models \mathcal{S}_\mathcal{Q} \qquad \mathcal{S}_\mathcal{Q} \sqsubseteq \mathcal{S}/\mathcal{S}_\mathcal{P}}{\mathcal{P} \parallel \mathcal{Q} \models \mathcal{S}}.$$

This rule, based on Theorem 2, differs in having the premise $\mathcal{S}_\mathcal{Q} \sqsubseteq \mathcal{S}/\mathcal{S}_\mathcal{P}$ in place of $\mathcal{S}_\mathcal{P} \parallel \mathcal{S}_\mathcal{Q} \sqsubseteq \mathcal{S}$. This substitution is permitted by the results of Theorem 6. The condition $\mathcal{A}^I_\mathcal{P} \cap \mathcal{A}^I_\mathcal{Q} = \emptyset$ is necessary in order to show that $\mathcal{S}_\mathcal{P} \parallel \mathcal{S}_\mathcal{Q} \sqsubseteq \mathcal{S}_\mathcal{P} \parallel (\mathcal{S}/\mathcal{S}_\mathcal{P})$, given the constraints on parallel compositionality, and the fact that $\mathcal{A}^I_{\mathcal{S}_\mathcal{P}}$ and $\mathcal{A}^I_{\mathcal{S}/\mathcal{S}_\mathcal{P}}$ are always disjoint.

4 A Printing Example

We illustrate our assume-guarantee framework on a simple example of component-based design for a system concerned with printing a document. The system as a whole is composed of a job scheduler, a printer controller and the physical printer itself. Intuitively, the scheduler decides when a print job can *start*, and expects to be informed when the job has $finish$ed. The controller, on the other hand, waits for the *start* signal from the scheduler, after which it instructs the printer to *print* the document, and awaits confirmation from the printer that the document has *printed*. At this stage, the controller will signal to the scheduler that the job has $finish$ed. The printer accepts a *print* command, after which it will start to print the document, and will signify when the document is *printed*.

We iteratively derive a design by successively applying AG rules and constructions. We start by making use of two specifications for the combined effect of the scheduler and printer controller:

1. Spec1: If the number of jobs sent to *print* is equal to or one greater than the number of jobs *printed*, then the number of job *start*s must be equal to or one greater than the number of requests sent to *print*.
2. Spec2: If the number of jobs sent to *print* is equal to or one greater than the number of jobs *printed*, then a job must be *printed* before it can be $finished$, and no two jobs can be consecutively $finished$ without a document being *printed* in between.

Spec1 and Spec2 can be represented by the AG specifications $\langle \mathcal{R}_{\mathsf{Spec}}, \mathcal{G}_{\mathsf{Spec1}} \rangle$ and $\langle \mathcal{R}_{\mathsf{Spec}}, \mathcal{G}_{\mathsf{Spec2}} \rangle$ respectively, where the assumptions and guarantees are depicted in Figure 1. For simplicity, we represent sets of traces by means of finite automata, and annotate states with an F to indicate that a trace becomes inconsistent. The combined effect of Spec1 and Spec2 is given by the conjunctive specification $\mathsf{Spec1} \wedge \mathsf{Spec2} = \langle \mathcal{R}_{\mathsf{Spec}}, \mathcal{G}_{\mathsf{Spec1} \wedge \mathsf{Spec2}} \rangle$, the guarantee of which is presented in Figure 2.

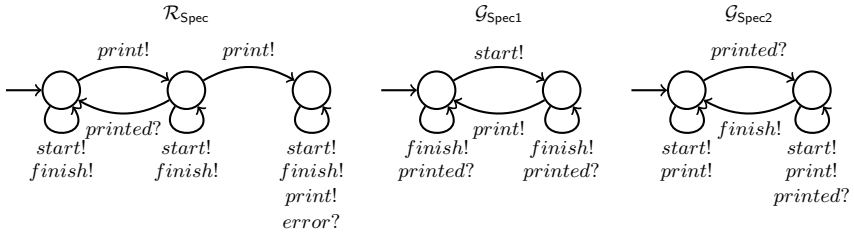


Fig. 1. Assumption and guarantees for Spec1 and Spec2

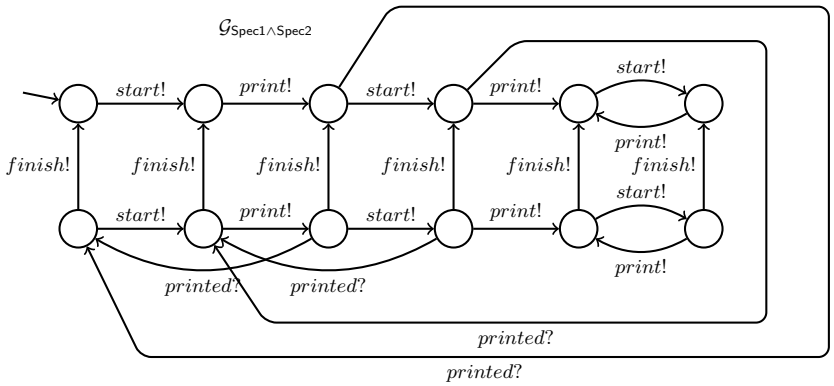


Fig. 2. The guarantee for Spec1 $\wedge$ Spec2

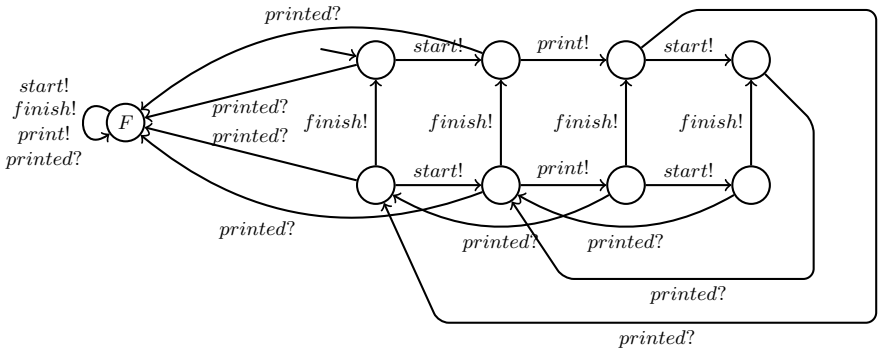


Fig. 3. The most general implementation of Spec1 $\wedge$ Spec2

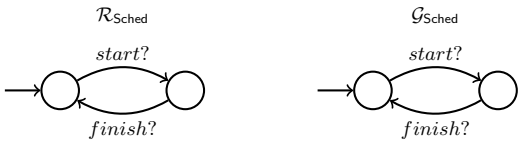


Fig. 4. Specification of a scheduling constraint Sched

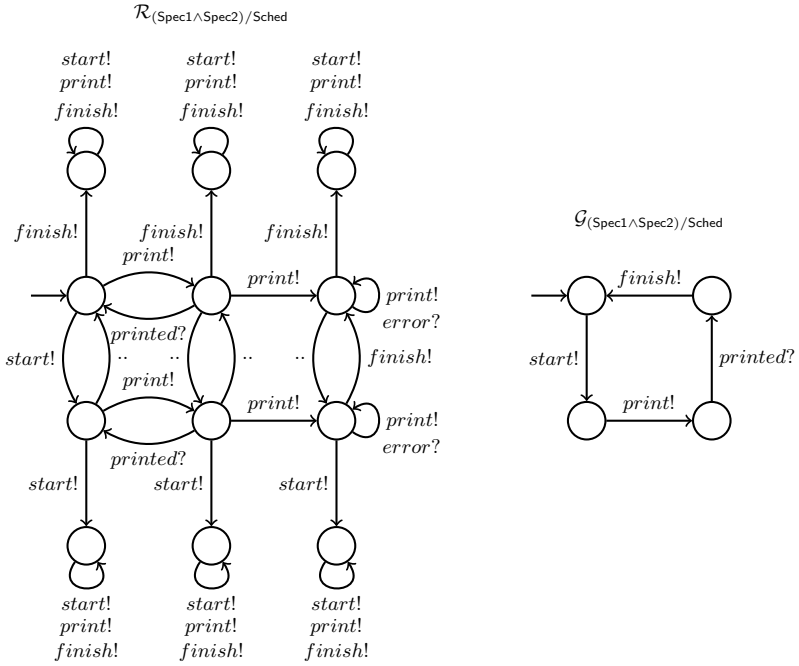


Fig. 5. Specification for $(\mathsf{Spec1} \wedge \mathsf{Spec2})/\mathsf{Sched}$

To demonstrate compositional AG reasoning, by Definition 10 we can find implementations $\mathcal{I}(\mathsf{Spec1})$ and $\mathcal{I}(\mathsf{Spec2})$ of Spec1 and Spec2 respectively, which by Theorem 5 allows us to derive $\mathcal{I}(\mathsf{Spec1}) \wedge \mathcal{I}(\mathsf{Spec2}) \models \mathsf{Spec1} \wedge \mathsf{Spec2}$. According to Lemma 4, this means that $\mathcal{I}(\mathsf{Spec1}) \wedge \mathcal{I}(\mathsf{Spec2}) \sqsubseteq_{imp} \mathcal{I}(\mathsf{Spec1} \wedge \mathsf{Spec2})$. Now by Theorem 3, we know $\mathsf{Spec1} \wedge \mathsf{Spec2} \sqsubseteq \mathsf{Spec1}$, so from Lemma 2 we obtain $\mathcal{I}(\mathsf{Spec1} \wedge \mathsf{Spec2}) \models \mathsf{Spec1}$, and from Lemma 4 we derive $\mathcal{I}(\mathsf{Spec1} \wedge \mathsf{Spec2}) \sqsubseteq_{imp} \mathcal{I}(\mathsf{Spec1})$. By similar reasoning it can be shown that $\mathcal{I}(\mathsf{Spec1} \wedge \mathsf{Spec2}) \sqsubseteq_{imp} \mathcal{I}(\mathsf{Spec2})$, hence by Theorem 2 of [4] we acquire $\mathcal{I}(\mathsf{Spec1} \wedge \mathsf{Spec2}) \sqsubseteq_{imp} \mathcal{I}(\mathsf{Spec1}) \wedge \mathcal{I}(\mathsf{Spec2})$. Mutual refinement of components in our framework corresponds to equality of models, so $\mathcal{I}(\mathsf{Spec1} \wedge \mathsf{Spec2}) = \mathcal{I}(\mathsf{Spec1}) \wedge \mathcal{I}(\mathsf{Spec2})$. Such an implementation is shown in Figure 3. Note how this component is unwilling to *print* after encountering two *start* requests not separated by a job being *printed*. This is because $\mathcal{R}_{\mathsf{Spec}}$ can issue an *error* after such an occurrence, but this is not ac-

cepted by $\mathcal{G}_{\mathsf{Spec1} \wedge \mathsf{Spec2}}$. Moreover, this implementation is able to *start* and *print* an unbounded number of jobs without ever having to $finish$ one of them.

We now propose an alternative derivation based on quotient, by making use of a constraint specification $\mathsf{Sched} = \langle \mathcal{R}_{\mathsf{Sched}}, \mathcal{G}_{\mathsf{Sched}} \rangle$ that requires *start* and $finish$ to alternate (shown in Figure 4). We wish to find an implementation for the printer controller, let it be called Controller, such that Controller is an implementation of $\mathsf{Spec1} \wedge \mathsf{Spec2}$ subject to the constraints imposed by Sched. This is equivalent to requiring $\mathsf{Controller} \models (\mathsf{Spec1} \wedge \mathsf{Spec2})/\mathsf{Sched}$. The specification $(\mathsf{Spec1} \wedge \mathsf{Spec2})/\mathsf{Sched}$ is exhibited in Figure 5, and the most general implementation is obtained from $\mathcal{G}_{(\mathsf{Spec1} \wedge \mathsf{Spec2})/\mathsf{Sched}}$ by appending all non-enabled inputs as inconsistent traces. In contrast to $\mathcal{I}(\mathsf{Spec1} \wedge \mathsf{Spec2})$, the constraints imposed by Sched on $\mathsf{Spec1} \wedge \mathsf{Spec2}$ means that any candidate implementation for Controller will ensure that there can be at most one outstanding job that has not $finish$ed.

5 Conclusion

We have presented a complete specification theory for reasoning about safety properties of component behaviours with an explicit separation of assumptions from guarantees. Our theory supports refinement based on traces, which relates specifications by implementation containment. We define compositional operations of parallel composition, as well as – for the first time in this setting – conjunction and quotient, directly on AG specifications. We give sound and complete AG reasoning rules for the three operators, which preserve safety and enable the reasoning about, e.g., safe substitutivity of components synthesised at run-time. The theory can be extended with disjunction and hiding, as well as liveness through the introduction of quiescence. The AG rules can also be fully automated, as they are based on simple set-theoretic operations and do not require the learning of assumptions. The refinement is linear-time, and hence amenable to automata-theoretic approaches.

Acknowledgments. The authors are supported by EU FP7 project CONNECT and ERC Advanced Grant VERIWARE.

References

1. Larsen, K.G., Nyman, U., Wasowski, A.: Modal I/O automata for interface and product line theories. In Nicola, R.D., ed.: ESOP. Volume 4421 of LNCS., Springer (2007) 64–79
2. Doyen, L., Henzinger, T.A., Jobstmann, B., Petrov, T.: Interface theories with component reuse. In: Proc. 8th ACM international conference on Embedded software. EMSOFT '08, ACM (2008) 79–88
3. Raclet, J.B., Badouel, E., Benveniste, A., Caillaud, B., Legay, A., Passerone, R.: A modal interface theory for component-based design. Fundam. Inform. **108** (2011) 119–149

4. Chen, T., Chilton, C., Jonsson, B., Kwiatkowska, M.: A Compositional Specification Theory for Component Behaviours. In Seidl, H., ed.: ESOP'12. Volume 7211 of LNCS., Springer (2012) 145–165
5. de Alfaro, L., Henzinger, T.A.: Interface automata. SIGSOFT Softw. Eng. Notes **26** (2001) 109–120
6. Bhaduri, P., Ramesh, S.: Interface synthesis and protocol conversion. Form. Asp. Comput. **20** (2008) 205–224
7. Abadi, M., Lamport, L.: Composing specifications. ACM Transactions on Programming Languages and Systems **15** (1993) 73–132
8. Abadi, M., Plotkin, G.: A logical view of composition. Theoretical Computer Science **114** (1993) 3–30
9. Pnueli, A.: Logics and models of concurrent systems. Springer (1985) 123–144
10. Clarke, E., Long, D., McMillan, K.: Compositional model checking. In: Proc. 4th Annual Symposium on Logic in computer science, IEEE Press (1989) 353–362
11. Grumberg, O., Long, D.E.: Model checking and modular verification. ACM Transactions on Programming Languages and Systems **16** (1991)
12. Maier, P.: A set-theoretic framework for assume-guarantee reasoning. In Orejas, F., Spirakis, P.G., Leeuwen, J., eds.: ICALP'01, LNCS 2076. (2001) 821–834
13. Namjoshi, K.S., Trefler, R.J.: On the completeness of compositional reasoning methods. ACM Trans. Comput. Logic **11** (2010) 16:1–16:22
14. Emmi, M., Giannakopoulou, D., Păsăreanu, C.: Assume-Guarantee Verification for Interface Automata. In Cuellar, J., Maibaum, T., Sere, K., eds.: FM 2008: Formal Methods. Volume 5014 of LNCS. Springer (2008) 116–131
15. Larsen, K.G., Nyman, U., Wasowski, A.: Interface input/output automata. In: FM 2006. Volume 4085 of LNCS., Springer (2006) 82–97
16. Bauer, S., David, A., Hennicker, R., Larsen, K., Legay, A., Nyman, U., Wasowski, A.: Moving from specifications to contracts in component-based design. In Lara, J., Zisman, A., eds.: FASE'12. Volume 7212 of LNCS. Springer (2012) 43–58
17. Delahaye, B., Caillaud, B., Legay, A.: Probabilistic contracts: a compositional reasoning methodology for the design of systems with stochastic and/or nondeterministic aspects. FMSD **38** (2011) 1–32
18. Chilton, C., Jonsson, B., Kwiatkowska, M.: Assume-Guarantee Reasoning for Safe Component Behaviours. Technical Report CS-RR-12-07, Department of Computer Science, University of Oxford (2012)
19. Collette, P.: Application of the composition principle to Unity-like specifications. In: TAPSOFT'93, LNCS 668, Springer-Verlag (1993) 230–242
20. Abadi, M., Lamport, L.: Conjoining specifications. ACM Transactions on Programming Languages and Systems **17** (1995) 507–534
21. Jonsson, B., Tsay, Y.K.: Assumption/guarantee specifications in linear-time temporal logic. Theoretical Computer Science **167** (1996) 47–72

A.2 Revisiting Timed Specification Theories: A Linear-Time Perspective

Revisiting Timed Specification Theories: A Linear-Time Perspective

Chris Chilton, Marta Kwiatkowska, and Xu Wang

Department of Computer Science, University of Oxford, UK

Abstract. We consider the setting of component-based design for real-time systems with critical timing constraints. Based on our earlier work, we propose a compositional specification theory for timed automata with I/O distinction, which supports substitutive refinement. Our theory provides the operations of parallel composition for composing components at run-time, logical conjunction/disjunction for independent development, and quotient for incremental synthesis. The key novelty of our timed theory lies in a weakest congruence preserving safety as well as bounded liveness properties. We show that the congruence can be characterised by two linear-time semantics, *timed-traces* and *timed-strategies*, the latter of which is derived from a game-based interpretation of timed interaction.

1 Introduction

Component-based design methodologies can be encapsulated in the form of compositional specification theories, which allow the mixing of specifications and implementations, admit substitutive refinement to facilitate reuse, and provide a rich collection of operators. Several such theories have been introduced in the literature, but none simultaneously address the following requirements: support for asynchronous input/output (I/O) communication with non-blocking outputs and non-input receptiveness; linear-time refinement preorder, so as to interface with automata and learning techniques; substitutivity of refinement, to allow for component reuse at runtime without introducing errors; and strong algebraic and compositionality properties, to enable offline as well as runtime reasoning.

Previously [1], we developed a linear-time specification theory for reasoning about untimed components that interact by synchronisation of I/O actions. Models can be specified operationally by means of transition systems augmented by an inconsistency predicate on states, or declaratively using traces. The theory admits non-determinism, a substitutive refinement preorder based on traces, and the operations of parallel composition, conjunction and quotient. The refinement is strictly weaker than alternating simulation and is actually the weakest pre-congruence preserving freeness of inconsistent states.

In this paper we target component-based development for real-time systems with critical timing constraints, such as embedded system components, the middleware layer and asynchronous hardware. Amongst notable works in the literature, we surveyed the theory of timed interfaces [2] and the theory of timed

specifications [3]. Though both support I/O distinctions, their refinement relations are not linear time: in [2], refinement (compatibility) is based on timed games, and in [3] it is a timed version of the alternating simulation originally defined for interface automata [4]. Consequently, it is too strong for determining when a component can be safely substituted for another. As an example, consider the transition systems P and Q in Figure 3: these should be equivalent in the sense of substitutivity under any environment, and are equivalent in our formulation (Definition 5), but they are not so according to timed alternating simulation.

Contributions. We formulate an elegant timed, asynchronous specification theory based on finite traces which supports substitutive refinement, as a timed extension of the linear-time specification theory of [1]. We allow for both operational descriptions of components, as well as declarative specifications based on traces. Our operational models are a variant of timed automata with I/O distinction (although we do not insist on input-enabledness, cf [5]), augmented by two special states: the *inconsistent* state $\bot$ represents safety and bounded-liveness errors, while the *timestop* state $\top$ is a novel addition representing either unrealisable output (if the component is not willing to produce that output) or unrealisable time-delay (if the delay would violate the invariant on that state).

Timestop models the ability to stop the clock and has been used before in embedded system and circuit design [6, 7]. It is notationally convenient, accounting for simpler definitions and a cleaner formalism. By enhancing the automata with the notion of *co-invariant*, we can, for the first time, distinguish the roles of input/output guards and invariant/co-invariants as specifying safety and bounded-liveness timed assumptions/guarantees. We emphasise that this is achieved with finite traces only; note that in the untimed case it would be necessary to extend to infinite traces to model liveness. In addition to *timed-trace semantics*, we present *timed-strategy semantics*, which coincides with the former but relates our work closer to the timed-game frameworks used by [3] and [2], and could in future serve as a guide to implementation of the theory. Finally, the *substitutive refinement* of our framework gives rise to the weakest congruence preserving $\bot$-freeness, which is not the case in the formalism of [3].

Related work. Our work can be seen as an alternative to the timed theories of [2, 3]. Being linear-time in spirit, it is also a generalisation of [8], an untimed theory inspired by asynchronous circuits, and Dill's trace theory [9]. The specification theory in [3] also introduces parallel, conjunction and quotient, but uses timed alternating simulation as refinement, which does not admit the weakest precongruence. An advantage of [3] is the algorithmic efficiency of branching-time simulation checking as well as the implementation reported in [10]. We briefly mention other related works, which include timed modal transition systems [11, 12], the timed I/O model [5, 13] and asynchronous circuits and embedded systems [14, 15]. A more detailed comparison based on the technical details of our work is included in Section 5. A full version of this paper including an even greater comparison with related work, in addition to proofs, is available as [16].

2 Formal Framework

In this section we introduce timed I/O automata, timed I/O transition systems and a semantic mapping from the former to the latter. Timed I/O automata are compact representations of timed I/O transition systems. We also present an operational specification theory based on timed I/O transition systems, which are endowed with a richer repertoire of semantic machinery than the automata.

2.1 Timed I/O Automata

Clock constraints. Given a set X of real-valued clock variables, a *clock constraint* over X, $cc : CC(X)$, is a boolean combination of atomic constraints of the form $x \bowtie d$ and $x - y \bowtie d$ where $x, y \in X$, $\bowtie \in \{\leq, <, =, >, \geq\}$, and $d \in \mathbb{N}$.

A *clock valuation* over X is a map t that assigns to each clock variable x in X a real value from $\mathbb{R}^{\geq 0}$. We say t satisfies cc, written $t \in cc$, if cc evaluates to true under valuation t. $t + d$ denotes the valuation derived from t by increasing the assigned value on each clock variable by $d \in \mathbb{R}^{\geq 0}$ time units. $t[rs \mapsto 0]$ denotes the valuation obtained from t by resetting the clock variables in rs to 0. Sometimes we use 0 for the clock valuation that maps all clock variables to 0.

Definition 1. *A* timed I/O automaton *(TIOA) is a tuple* $(C, I, O, L, l^0, AT, Inv, coInv)$*, where:*

- $C \subseteq X$ *is a finite set of clock variables*
- $A\ (= I \uplus O)$ *is a finite alphabet, consisting of inputs* I *and outputs* O
- L *is a finite set of* locations *and* $l^0 \in L$ *is the* initial location
- $AT \subseteq L \times CC(C) \times A \times 2^C \times L$ *is a set of* action transitions
- $Inv : L \to CC(C)$ *and* $coInv : L \to CC(C)$ *assign* invariants *and* co-invariants *to states, each of which is a downward-closed clock constraint.*

We use l, l', l_i to range over L and use $l \xrightarrow{g,a,rs} l'$ as a shorthand for $(l, g, a, rs, l') \in AT$. $g : CC(C)$ is the enabling guard of the transition, $a \in A$ the action, and rs the subset of clock variables to be reset.

Our TIOAs are timed automata that distinguish input from output and invariant from co-invariant. They are similar to existing variants of timed automata with input/output distinction, except for the introduction of co-invariants and non-insistence on input-enabledness. While invariants specify the bounds beyond which time may not progress, co-invariants specify the bounds beyond which the system will *time-out* and enter error states. It is designed for the assume/guarantee specification of timed components, in order to specify both the assumptions made by the component on the inputs and the guarantees provided by the component on the outputs, with respect to timing constraints.

Guards on output transitions express *safety timing guarantees*, while guards on input transitions express *safety timing assumptions*. On the other hand, invariants (urgency) express *liveness timing guarantees* on the outputs at the locations they decorate, while co-invariants (time-out) express *liveness timing assumptions* on the inputs at those locations.

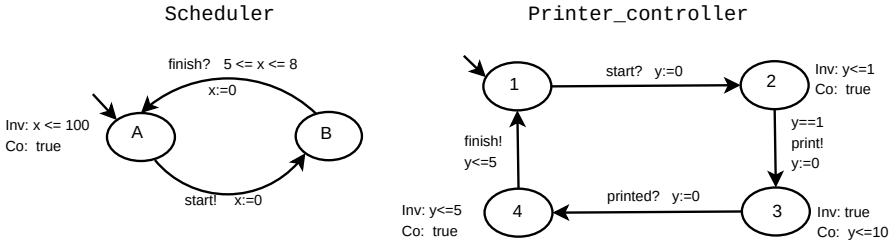


Fig. 1. Job scheduler and printer controller.

When two components are composed, the parallel composition automatically checks whether the guarantees provided by one component meet the assumptions required by the other. For instance, the unexpected arrival of an input at a particular location and time (indicated by a non-enabled transition) leads to a *safety error* in the parallel composition. The non-arrival of an expected input at a location before its time-out (specified by the co-invariant) leads to a *bounded-liveness error* in the parallel composition.

Example. Figure 1 depicts TIOAs representing a job scheduler together with a printer controller. The invariant at location A of the scheduler forces a bounded-liveness guarantee on outputs in that location. As time must be allowed to progress beyond $t = 100$, the *start* action must be fired within the range $0 \leq t \leq 100$. After *start* has been fired, the clock x is reset to 0 and the scheduler waits (possibly indefinitely) for the job to *finish*. If the job does finish, the scheduler is only willing for this to take place between $5 \leq t \leq 8$ after the job started (safety assumption), otherwise an unexpected input error will be thrown.

The controller waits for the job to *start*, after which it will wait exactly 1 time unit before issuing *print* (forced by the invariant $y \leq 1$ on state 2 and the guard $y = 1$). The controller now requires the printer to indicate the job is *printed* within 10 time units of being sent to the printer, otherwise a time-out error on inputs will occur (co-invariant $y \leq 10$ in state 3 as liveness assumption). After the job has finished printing, the controller must indicate to the scheduler that the job has *finish*ed within 5 time units.

Notation. For a set of input actions I and a set of output actions O, define $tA = I \uplus O \uplus \mathbb{R}^{>0}$ to be the set of *timed actions*, $tI = I \uplus \mathbb{R}^{>0}$ to be the set of *timed inputs*, and $tO = O \uplus \mathbb{R}^{>0}$ to be the set of *timed outputs*. We use symbols like α, β, etc. to range over tA.

A *timed word* (ranged over by w, w', w_i etc.) is a finite mixed sequence of positive real numbers ($\mathbb{R}^{>0}$) and visible actions such that *no two numbers are adjacent to one another*. For instance, $\langle 0.33, a, 1.41, b, c, 3.1415\rangle$ is a timed word denoting the observation that action a occurs at 0.33 time units, then another 1.41 time units lapse before the simultaneous occurrence of b and c, which is followed by 3.1415 time units of no event occurrence. ϵ denotes the empty word.

Concatenation of timed words w and w' is obtained by appending w' onto the end of w and coalescing adjacent reals (summing them). Prefix/extension

are defined as usual by concatenation. We write $w \restriction tA_0$ for the projection of w onto timed alphabet tA_0, which is defined by removing from w all actions not inside tA_0 and coalescing adjacent reals.

2.2 Semantics as Timed I/O Transition Systems

The semantics of TIOAs are given as timed I/O transition systems, which are a special class of infinite labelled transition systems.

Definition 2. *A* timed I/O transition system *(TIOTS) is a tuple* $\mathcal{P} = \langle I, O, S, s^0, \rightarrow \rangle$*, where* I *and* O *are the input and output actions respectively,* $S = (L \times \mathbb{R}^C) \uplus \{\bot, \top\}$ *is a set of states,* $s^0 \in S$ *is the designated initial state, and* $\rightarrow \subseteq S \times (I \uplus O \uplus \mathbb{R}^{>0}) \times S$ *is the action and time-labelled transition relation.*

The states of the TIOTS for a TIOA capture the configuration of the automaton, i.e. its location and clock valuation. Therefore, each state of the TIOTS is a pair drawn from $L \times \mathbb{R}^C$, which we refer to as the set of *plain states*. In addition, we introduce two special states $\bot$ and $\top$, which are required for the semantic mapping of disabled inputs/outputs, invariants and co-invariants. In the rest of the paper, we use p, p', p_i to range over $P = L \times \mathbb{R}^C$ while s, s', s_i range over S.

$\bot$ is the so-called *inconsistent state*, arising through assumption/guarantee mismatches, i.e. safety and bounded-liveness errors. $\top$ is the so-called *timestop state*, representing the *magic moment* from which time stops elapsing and no error can occur. We assume that $\top$ refines plain states, which in turn refine $\bot$. For technical convenience (e.g. ease of defining time additivity and trace semantics), we require that $\top$ and $\bot$ are a *chaotic states*, i.e. states having self-loops for each $\alpha \in tA$.

On TIOTSs, a disabled input in a state p is equated to an input transition from p to $\bot$, while a disabled output/delay in p is equated to an output/delay from p to $\top$. The intuition here comes from the I/O game perspective. The component controls output and delay, while the environment controls input. $\bot$ is the losing state for the environment, so an input transition from p to $\bot$ is a transition that the environment tries to avoid at all cost (unless there is no choice). $\top$ is the losing state for the component, so an output/delay transition from p to $\top$ is a transition that the component tries to avoid at any cost. Thus we can have two semantic-preserving transformations on TIOTSs.

The $\bot$-*completion* of a TIOTS $\mathcal{P}$, denoted $\mathcal{P}^\bot$, adds an a-labelled transition from p to $\bot$ for every $p \in P$ $(= L \times \mathbb{R}^C)$ and $a \in I$ s.t. a is not enabled at p.[1] The $\top$-*completion*, denoted $\mathcal{P}^\top$, adds an α-labelled transition from p to $\top$ for every $p \in P$ and $\alpha \in tO$ s.t. α is not enabled at p.

Now, the transition relation $\rightarrow$ of the TIOTS is derived from the execution semantics of the TIOA.

Definition 3. *Let* $\mathcal{P}$ *be a TIOA. The execution semantics of* $\mathcal{P}$ *is a TIOTS* $\langle I, O, S, s^0, \rightarrow \rangle$*, where:*

[1] $\bot$-completion will make a TIOTS *input-receptive*, i.e. input-enabled in all states.

- $S = (L \times \mathbb{R}^C) \uplus \{\bot, \top\}$
- $s^0 = \top$ *providing* $0 \notin Inv(l^0)$, $s^0 = \bot$ *providing* $0 \in Inv(l^0) \wedge \neg coInv(l^0)$ *and* $s^0 = (l^0, 0)$ *providing* $0 \in Inv(l^0) \wedge coInv(l^0)$,
- $\rightarrow$ *is the smallest relation satisfying:*
  1. *If* $l \xrightarrow{g,a,rs} l'$, $t' = t[rs \mapsto 0]$, $t \in Inv(l) \wedge coInv(l) \wedge g$, *then:*
     (a) plain action: $(l,t) \xrightarrow{a} (l',t')$ *providing* $t' \in Inv(l') \wedge coInv(l')$
     (b) error action: $(l,t) \xrightarrow{a} \bot$ *providing* $t' \in Inv(l') \wedge \neg coInv(l')$
     (c) magic action: $(l,t) \xrightarrow{a} \top$ *providing* $t' \in \neg Inv(l')$ *and* $a \in I$.
  2. plain delay: $(l,t) \xrightarrow{d} (l, t+d)$ *if* $t, t+d \in Inv(l) \wedge coInv(l)$
  3. time-out delay: $(l,t) \xrightarrow{d} \bot$ *if* $t \in Inv(l) \wedge coInv(l)$, $t + d \notin coInv(l)$ *and* $\exists\, 0 < \delta \leq d : t + \delta \in Inv(l) \wedge \neg coInv(l)$.

Note that our semantics tries to minimise the use of transitions leading to $\top/\bot$ states. Thus there are no delay or output transitions leading to $\top$. However, there are *implicit timestops*, which we capture using the concept of *semi-timestop* (i.e. semi-$\top$). We say a plain state p is a *semi-*$\top$ iff 1) all output transitions enabled in p and all of its time-passing successors lead to the $\top$ state, and 2) there exists $d \in \mathbb{R}^{>0}$ s.t. $p \xrightarrow{d} \top$ or d is not enabled in p. Thus a semi-$\top$ is a state in which it is impossible for the component to avoid the timestop without suitable inputs from the environment.

The introduction of timestop ($\top$), which can model the operation of stopping the system clock, is an unconventional aspect of our semantics. Certain real-world systems have an inherent ability to stop the clock, e.g. [6, 7], which are related to embedded systems and circuit design. When the suspension of clocks is not meaningful, it is necessary to remove timestop in order to leave the so-called realisable behaviour. Timestop is useful even for timestop free systems, as it can significantly simplify operations, such as quotient and conjunction.

TIOTS terminology. We say a TIOTS is *deterministic* iff $s \xrightarrow{\alpha} s' \wedge s \xrightarrow{\alpha} s''$ implies $s' = s''$, and is *time additive* providing $p \xrightarrow{d_1+d_2} s'$ iff $p \xrightarrow{d_1} s$ and $s \xrightarrow{d_2} s'$ for some s. In the sequel, we only consider time-additive TIOTSs.

Given a TIOTS $\mathcal{P}$, a timed word can be derived from a finite execution of $\mathcal{P}$ by extracting the labels in each transition and coalescing adjacent reals. The timed words derived from such executions are called *traces* of $\mathcal{P}$. We use tt, tt', tt_i to range over traces and write $s^0 \stackrel{tt}{\Rightarrow} s$ to denote a finite execution producing tt and leading to s.

2.3 Operational Specification Theory

In this section we develop a compositional specification theory for TIOTSs based on the operations of parallel composition $\|$, conjunction $\wedge$, disjunction $\vee$ and quotient $\%$. The operators are defined via transition rules that are a variant on synchronised product.

Parallel composition yields a TIOTS that represents the combined effect of its operands interacting with one another. The remaining operations must

Table 1. State representations under composition operators.

$\|\|$	$\top$	p_0	$\bot$
$\top$	$\top$	$\top$	$\top$
p_1	$\top$	$p_0 \times p_1$	$\bot$
$\bot$	$\top$	$\bot$	$\bot$

$\wedge$	$\top$	p_0	$\bot$
$\top$	$\top$	$\top$	$\top$
p_1	$\top$	$p_0 \times p_1$	p_1
$\bot$	$\top$	p_0	$\bot$

$\vee$	$\top$	p_0	$\bot$
$\top$	$\top$	p_0	$\bot$
p_1	p_1	$p_0 \times p_1$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$

$\%$	$\top$	p_0	$\bot$
$\top$	$\bot$	$\bot$	$\bot$
p_1	$\top$	$p_0 \times p_1$	$\bot$
$\bot$	$\top$	$\top$	$\bot$

be explained with respect to a refinement relation, which corresponds to safe-substitutivity in our theory. A TIOTS is a refinement of another if it will work in any environment that the original worked in without introducing safety or bounded-liveness errors. Conjunction yields the coarsest TIOTS that is a refinement of its operands, while disjunction yields the finest TIOTS that is refined by both of its operands. The operators are thus equivalent to the join and meet operations on TIOTSs[2]. Quotient is the adjoint of parallel composition, meaning that $\mathcal{P}_0 \% \mathcal{P}_1$ is the coarsest TIOTS such that $(\mathcal{P}_0 \% \mathcal{P}_1) \| \mathcal{P}_1$ is a refinement of $\mathcal{P}_0$.

Let $\mathcal{P}_i = \langle I_i, O_i, S_i, s_i^0, \rightarrow_i \rangle$ for $i \in \{0, 1\}$ be two TIOTSs that are both $\bot$ and $\top$-completed, satisfying (wlog) $S_0 \cap S_1 = \{\bot, \top\}$. The composition of $\mathcal{P}_0$ and $\mathcal{P}_1$ under the operation $\otimes \in \{\|, \wedge, \vee, \%\}$, written $\mathcal{P}_0 \otimes \mathcal{P}_1$, is only defined when certain *composability* restrictions are imposed on the alphabets of the TIOTSs. $\mathcal{P}_0 \parallel \mathcal{P}_1$ is only defined when the output sets of $\mathcal{P}_0$ and $\mathcal{P}_1$ are disjoint, because an output should be controlled by at most one component. Conjunction and disjunction are only defined when the TIOTSs have *identical alphabets* (i.e. $O_0 = O_1$ and $I_0 = I_1$). This restriction can be relaxed at the expense of more cumbersome notation, which is why we focus on the simpler case in this paper. For the quotient, we require that the alphabet of $\mathcal{P}_0$ *dominates* that of $\mathcal{P}_1$ (i.e. $A_1 \subseteq A_0$ and $O_1 \subseteq O_0$), in addition to $\mathcal{P}_1$ being a deterministic TIOTS. As quotient is a synthesis operator, it is difficult to give a definition using just *state-local* transition rules, since quotient needs global information about the transition systems. This is why we insist on $\mathcal{P}_1$ being deterministic[3].

Definition 4. *Let $\mathcal{P}_0$ and $\mathcal{P}_1$ be TIOTSs composable under $\otimes \in \{\|, \wedge, \vee, \%\}$. Then $\mathcal{P}_0 \otimes \mathcal{P}_1 = \langle I, O, S, s^0, \rightarrow \rangle$ is the TIOTS where:*

- *If $\otimes = \|$, then $I = (I_0 \cup I_1) \setminus O$ and $O = O_0 \cup O_1$*
- *If $\otimes \in \{\wedge, \vee\}$, then $I = I_0 = I_1$ and $O = O_0 = O_1$*
- *If $\otimes = \%$, then $I = I_0 \cup O_1$ and $O = O_0 \setminus O_1$*
- *$S = (P_0 \times P_1) \uplus P_0 \uplus P_1 \uplus \{\top, \bot\}$*
- *$s^0 = s_0^0 \otimes s_1^0$*
- *$\rightarrow$ is the smallest relation containing $\rightarrow_0 \cup \rightarrow_1$, and satisfying the rules:*

$$\frac{p_0 \xrightarrow{\alpha}_0 s_0' \quad p_1 \xrightarrow{\alpha}_1 s_1'}{p_0 \otimes p_1 \xrightarrow{\alpha} s_0' \otimes s_1'} \qquad \frac{p_0 \xrightarrow{a}_0 s_0' \quad a \notin A_1}{p_0 \otimes p_1 \xrightarrow{a} s_0' \otimes p_1} \qquad \frac{p_1 \xrightarrow{a}_0 s_1' \quad a \notin A_0}{p_0 \otimes p_1 \xrightarrow{a} p_0 \otimes s_1'}$$

[2] As we write $A \sqsubseteq B$ to mean A is refined by B, our operators $\wedge$ and $\vee$ are reversed in comparison to the standard symbols for meet and join.

[3] Technically speaking, the problem is a consequence of state quotient being right-distributive but not left-distributive over state disjunction (cf Table 1).

We adopt the notation of $s_0 \otimes s_1$ for states, where the associated interpretation is supplied in Table 1. Furthermore, given two plain states $p_i = (l_i, t_i)$ for $i \in \{0, 1\}$, we define $p_0 \times p_1 = ((l_0, l_1), t_0 \uplus t_1)$.

Table 1 tells us how states should be combined under the composition operators. For parallel, a state is magic if one component state is magic, and a state is error if one component is error while the other is not magic. For conjunction, encountering error in one component implies the component can be discarded and the rest of the composition behaves like the other component. The conjunction table follows the intuition of the join operation on the refinement preorder. Similarly for disjunction. Quotient is the adjoint of parallel composition. If the second component state does not refine the first, the quotient will try to rescue the refinement by producing $\top$ (so that its composition with the second will refine the first). If the second component state does refine the first, the quotient will produce the least refined value so that its composition with the second will not break the refinement.

An *environment* for a TIOTS $\mathcal{P}$ is any TIOTS $\mathcal{Q}$ such that the alphabet of $\mathcal{Q}$ is *complementary* to that of $\mathcal{P}$, meaning $I_\mathcal{P} = O_\mathcal{Q}$ and $O_\mathcal{P} = I_\mathcal{Q}$. Refinement in our framework corresponds to contextual substitutability, in which the context is an arbitrary environment.

Definition 5. *Let $\mathcal{P}_{imp}$ and $\mathcal{P}_{spec}$ be TIOTSs with identical alphabets. $\mathcal{P}_{imp}$* refines *$\mathcal{P}_{spec}$, denoted $\mathcal{P}_{spec} \sqsubseteq \mathcal{P}_{imp}$, iff for all environments $\mathcal{Q}$, $\mathcal{P}_{spec} \parallel \mathcal{Q}$ is $\perp$-free implies $\mathcal{P}_{imp} \parallel \mathcal{Q}$ is $\perp$-free. We say $\mathcal{P}_{imp}$ and $\mathcal{P}_{spec}$ are* substitutively equivalent*, i.e. $\mathcal{P}_{spec} \simeq \mathcal{P}_{imp}$, iff $\mathcal{P}_{imp} \sqsubseteq \mathcal{P}_{spec}$ and $\mathcal{P}_{spec} \sqsubseteq \mathcal{P}_{imp}$.*

It is obvious that $\simeq$ induces the weakest equivalence on TIOTSs that preserves $\perp$-freeness. In the sequel, we give two concrete characterisations of $\simeq$ and show it to be a congruence w.r.t. the operators of the specification theory.

The operational definition of quotient requires $\mathcal{P}_1$ to be deterministic. For any TIOTS $\mathcal{P}$, a semantically-equivalent deterministic component can be obtained, denoted $\mathcal{P}^D$, by means of a modified subset construction acting on $(\mathcal{P}^\perp)^\top$. For any subset S_0 of states reachable by a given trace, we only keep those which are minimal w.r.t. the state refinement relation. So if the current state subset S_0 contains $\perp$, the procedure reduces S_0 to $\perp$; if $\perp \notin S_0 \neq \{\top\}$, it reduces S_0 by removing any potential $\top$ in S_0.[4]

Proposition 1. *For any TIOTS $\mathcal{P}$, it holds that $\mathcal{P} \simeq \mathcal{P}^D$.*

Equipped with determinisation, quotient is a fully defined operator on any pair of TIOTSs. Furthermore, we can give an alternative (although substitutively equivalent) formulation of quotient as the derived operator $(\mathcal{P}_0^\neg \parallel \mathcal{P}_1)^\neg$, where $\neg$ is a mirroring operation that first determinises its argument, then interchanges the input and output sets, as well as the $\top$ and $\perp$ states.

[4] A detailed definition of transforming untimed non-deterministic systems into substitutively-equivalent deterministic ones is contained in Definition 4.2 of [8].

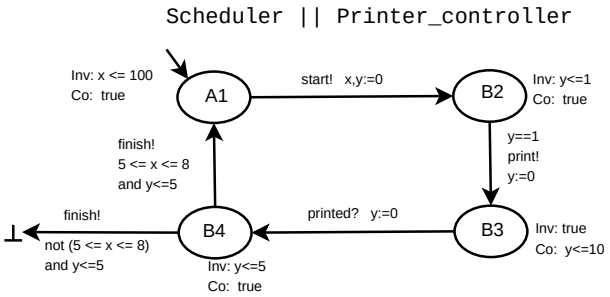


Fig. 2. Parallel composition of the job scheduler and printer controller.

Example. Figure 2 shows the parallel composition of the job scheduler with the printer controller. In the transition from $B4$ to $A1$, the guard combines the effects of the constraints on the clocks x and y. As *finish* is an output of the controller, it can be fired at a time when the scheduler is not expecting it, meaning that a safety error will occur. This is indicated by the transition to $\bot$ when the guard constraint $5 \leq x \leq 8$ is not satisfied.

3 Timed I/O Game

Our specification theory can be seen as an I/O game between a *component* and an *environment* that uses a *coin* to break ties. The specification of a component (in the form of a TIOA or TIOTS) is built to encode the set of strategies possible for the component in the game (just like an NFA encodes a set of words).

- Given two TIOTSs $\mathcal{P}$ and $\mathcal{Q}$ with identical alphabets, we say $\mathcal{P}$ is a partial unfolding [17] of $\mathcal{Q}$ if there exists a function f from $S_\mathcal{P}$ to $S_\mathcal{Q}$ s.t. 1) f maps $\top$ to $\top$, $\bot$ to $\bot$, and plain states to plain states, 2) $f(s^0_\mathcal{P}) = s^0_\mathcal{Q}$, and 3) $p \xrightarrow{\alpha}_\mathcal{P} s \Rightarrow f(p) \xrightarrow{\alpha}_\mathcal{Q} f(s)$.
- We say an acyclic TIOTS is a *tree* if 1) there does not exist a pair of transitions in the form of $p \xrightarrow{a} p''$ and $p' \xrightarrow{d} p''$, 2) $p \xrightarrow{a} p'' \wedge p' \xrightarrow{b} p''$ implies $p = p'$ and $a = b$ and 3) $p \xrightarrow{d} p'' \wedge p' \xrightarrow{d} p''$ implies $p = p'$.
- We say an acyclic TIOTS is a *simple path* if 1) $p \xrightarrow{a} s' \wedge p \xrightarrow{\alpha} s''$ implies $s' = s''$ and $a = \alpha$ and 2) $p \xrightarrow{d} s' \wedge p \xrightarrow{d} s''$ implies $s' = s''$.
- We say a simple path $\mathcal{L}$ is a *run* of $\mathcal{P}$ if $\mathcal{L}$ is a partial unfolding of $\mathcal{P}$.

Strategies. A *strategy* $\mathcal{G}$ is a deterministic tree TIOTS s.t. each plain state in $\mathcal{G}$ is ready to accept all possible inputs by the environment, but allows a single move (delay or output) by the component, i.e. $eb_\mathcal{G}(p) = I \uplus mv_\mathcal{G}(p)$ s.t. $mv_\mathcal{G}(p) = \{a\}$ for some $a \in O$ or $mv_\mathcal{G}(p) \subseteq \mathbb{R}^{>0}$, where $eb_\mathcal{G}(p)$ denotes the set of enabled timed actions in state p of LTS $\mathcal{G}$, and $mv_\mathcal{G}(p)$ denotes the unique component move allowed by $\mathcal{G}$ at p.

A TIOTS $\mathcal{P}$ *contains* a strategy $\mathcal{G}$ if $\mathcal{G}$ is a partial unfolding of $(\mathcal{P}^\bot)^\top$. The set of strategies contained in $\mathcal{P}$ is denoted $stg(\mathcal{P})$. Since it makes little sense to

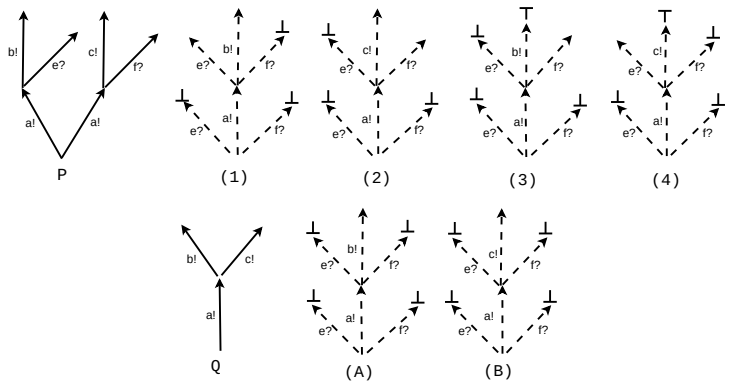


Fig. 3. Strategy example.

distinguish strategies that are isomorphic, we will freely use strategies to refer to their isomorphism classes and write $\mathcal{G} = \mathcal{G}'$ to mean $\mathcal{G}$ and $\mathcal{G}'$ are isomorphic.

Figure 3 illustrates the idea of strategies. For simplicity, we use two untimed transition systems $\mathcal{P}$ and $\mathcal{Q}$ with identical alphabets $I = \{e, f\}$ and $O = \{a, b, c\}$. The transition systems use solid lines, while strategies use dotted lines. Plain states are unmarked, while the $\top$ and $\bot$ states are labelled as such[5]. A subset of the strategies for $\mathcal{P}$ and $\mathcal{Q}$ are shown on the right hand side of the respective components. Note that strategies 3 and 4 arise through $\top$-completion.

Comparing strategies. When the game is played, the component tries to avoid reaching $\top$, while the environment tries to avoid reaching $\bot$. Strategies in $stg(\mathcal{P})$ vary in their effectiveness to achieve this objective, which induces a hierarchy on strategies that closely resemble one another. We say $\mathcal{G}$ and $\mathcal{G}'$ are *affine* if $s^0_{\mathcal{G}} \overset{tt}{\Rightarrow} p$ and $s^0_{\mathcal{G}'} \overset{tt}{\Rightarrow} p'$ implies $mv_{\mathcal{G}}(p) = mv_{\mathcal{G}'}(p')$. Intuitively, it means $\mathcal{G}$ and $\mathcal{G}'$ propose the same move at the 'same' states. For instance, the strategies 1, 3 and A in Figure 3 are pairwise affine and so are the strategies 2, 4 and B.

Given two affine strategies $\mathcal{G}$ and $\mathcal{G}'$, we say $\mathcal{G}$ is *more aggressive* than $\mathcal{G}'$, denoted $\mathcal{G} \preceq \mathcal{G}'$, if 1) $s^0_{\mathcal{G}'} \overset{tt}{\Rightarrow} \bot$ implies there is a prefix tt_0 of tt s.t. $s^0_{\mathcal{G}} \overset{tt_0}{\Longrightarrow} \bot$ and 2) $s^0_{\mathcal{G}} \overset{tt}{\Rightarrow} \top$ implies there is a prefix tt_0 of tt s.t. $s^0_{\mathcal{G}'} \overset{tt_0}{\Longrightarrow} \top$. Intuitively, it means $\mathcal{G}$ can reach $\bot$ faster but $\top$ slower than $\mathcal{G}'$. $\preceq$ forms a partial order over $stg(\mathcal{P})$, or more generally, over any set of strategies with identical alphabets. For instance, strategy A is more aggressive than 1 and 3, while strategy B is more aggressive than 2 and 4.

When the game is played, the component $\mathcal{P}$ prefers to use the maximally aggressive strategies in $stg(\mathcal{P})$[6]. Thus two components that differ only in non-maximally aggressive strategies should be equated. We define the *strategy semantics* of component $\mathcal{P}$ to be $[\mathcal{P}]_s = \{\mathcal{G}' \mid \exists \mathcal{G} \in stg(\mathcal{P}) : \mathcal{G} \preceq \mathcal{G}'\}$, i.e. the upward-closure of $stg(\mathcal{P})$ w.r.t. $\preceq$.

[5] For simplicity, we allow multiple copies of $\top$ and $\bot$, which are assumed to be chaotic.
[6] This is because our semantics is designed to preserve $\bot$ rather than $\top$.

Game rules. When a component strategy $\mathcal{G}$ is played against an environment strategy $\mathcal{G}'$, at each game state (i.e. a product state $p_\mathcal{G} \times p_{\mathcal{G}'}$) $\mathcal{G}$ and $\mathcal{G}'$ each propose a move (i.e. $mv_\mathcal{G}(p_\mathcal{G})$ and $mv_{\mathcal{G}'}(p_{\mathcal{G}'})$). If one of them is a delay and the other is an action, the action will prevail. If both propose delay moves (i.e. $mv_\mathcal{G}(p_\mathcal{G}), mv_{\mathcal{G}'}(p_{\mathcal{G}'}) \subseteq \mathbb{R}^{>0}$), the smaller one (w.r.t. set containment) will prevail.[7]

Since a delay move proposed at a strategy state is the maximal set of possible delays enabled at that state, the next move proposed at the new state after firing the set must be an action move (due to time additivity). Thus a play cannot have two consecutive delay moves.

If, however, both propose action moves, there will be a tie, which will be resolved by tossing the coin. For uniformity's sake, the coin can be treated as a special component. A strategy of the coin is a function h from tA^* to $\{0, 1\}$. We denote the set of all possible coin strategies as H.

A play of the game can be formalised as a composition of three strategies, one each from the component, environment and coin, denoted $\mathcal{G}_\mathcal{P} \parallel_h \mathcal{G}_\mathcal{Q}$. At a current game state $p_\mathcal{P} \times p_\mathcal{Q}$, if the prevailing action is α and we have $p_\mathcal{P} \xrightarrow{\alpha} s'_\mathcal{P}$ and $p_\mathcal{Q} \xrightarrow{\alpha} s'_\mathcal{Q}$, then the next game state is $s_\mathcal{P} \parallel s_\mathcal{Q}$. The play will stop when it reaches either $\top$ or $\bot$. The composition will produce a simple path $\mathcal{L}$ that is a run of $\mathcal{P} \parallel \mathcal{Q}$. Since $\mathcal{P} \parallel \mathcal{Q}$ gives rise to a *closed system* (i.e. the input alphabet is empty), a run of $\mathcal{P} \parallel \mathcal{Q}$ is a strategy of $\mathcal{P} \parallel \mathcal{Q}$.

Thus, strategy composition of $\mathcal{P}$ and $\mathcal{Q}$ is closely related to their parallel composition: $stg(\mathcal{P} \parallel \mathcal{Q}) = \{\mathcal{G}_\mathcal{P} \parallel_h \mathcal{G}_\mathcal{Q} \mid \mathcal{G}_\mathcal{P} \in stg(\mathcal{P}), \mathcal{G}_\mathcal{Q} \in stg(\mathcal{Q}) \text{ and } h \in H\}$.

Parallel composition. Strategy composition, like component parallel composition, can be generalised to any pair of components $\mathcal{P}$ and $\mathcal{Q}$ with *composable alphabets*. That is, $O_\mathcal{P} \cap O_\mathcal{Q} = \{\}$. For such $\mathcal{P}$ and $\mathcal{Q}$, $\mathcal{G}_\mathcal{P} \parallel_h \mathcal{G}_\mathcal{Q}$ gives rise to a tree rather than a simple path TIOTS. That is, at each game state $p_\mathcal{P} \times p_\mathcal{Q}$, besides firing the prevailing $\alpha \in tO_\mathcal{P} \cup tO_\mathcal{Q}$, we need also to fire 1) all the synchronised inputs, i.e. $e \in I_\mathcal{P} \cap I_\mathcal{Q}$, and reach the new game state $s_\mathcal{P} \parallel s_\mathcal{Q}$ (assuming $p_\mathcal{P} \xrightarrow{e} s_\mathcal{P}$ and $p_\mathcal{Q} \xrightarrow{e} s_\mathcal{Q}$) and 2) all the independent inputs, i.e. $e \in (I_\mathcal{P} \cup I_\mathcal{Q}) \setminus (A_\mathcal{P} \cap A_\mathcal{Q})$, and reach the new game state $s_\mathcal{P} \times p_\mathcal{Q}$ or $p_\mathcal{P} \times s_\mathcal{Q}$. It is easy to verify that $\mathcal{G}_\mathcal{P} \parallel_h \mathcal{G}_\mathcal{Q}$ is a strategy of $\mathcal{P} \parallel \mathcal{Q}$.

Conjunction/disjunction. Strategy conjunction (&) and *strategy disjunction* (+) are binary operators defined only on pairs of affine strategies, by $\mathcal{G}\&\mathcal{G}' = \mathcal{G} \wedge \mathcal{G}'$ and $\mathcal{G} + \mathcal{G}' = \mathcal{G} \vee \mathcal{G}'$. If $\mathcal{G}$ and $\mathcal{G}'$ are not affine, $\mathcal{G} \wedge \mathcal{G}'$ and $\mathcal{G} \vee \mathcal{G}'$ may not produce a strategy. From Figure 3, the disjunction of strategies 1 and 2 will produce a transition system that stops to output after the a transition.

Refinement. Equality of strategies induces an equivalence on TIOTSs: $\mathcal{P}$ and $\mathcal{Q}$ are *strategy equivalent* iff $[\mathcal{P}]_s = [\mathcal{Q}]_s$. However, strategy equivalence is too fine for the purpose of *substitutive refinement* (cf Definition 5). For instance,

[7] Note that all invariants and co-invariants are downward-closed. Thus a delay move can be respresented as a time interval from 0 to some $d \in \mathbb{R}^{\geq 0}$.

transition systems $\mathcal{P}$ and $\mathcal{Q}$ in Figure 3 are substitutively equivalent, but are not strategy equivalent, because 1, 2, 3 and 4 are strategies of $\mathcal{Q}$ (due to upward-closure w.r.t. $\preceq$), while A and B are not strategies of $\mathcal{P}$.

However, we demonstrate that *substitutive equivalence is reducible to strategy equivalence* providing we perform *disjunction closure* on strategies.

Lemma 1. *Given a pair of affine component strategies $\mathcal{G}_0$ and $\mathcal{G}_1$, $\mathcal{G}_0 \parallel_h \mathcal{G}$ and $\mathcal{G}_1 \parallel_h \mathcal{G}$ are $\bot$-free for a pair of environment and coin strategies $\mathcal{G}$ and h iff $\mathcal{G}_0 + \mathcal{G}_1 \parallel_h \mathcal{G}$ is $\bot$-free.*

We say Π^+ is a *disjunction closure* of set of strategies Π iff it is the least superset of Π s.t. $\mathcal{G} + \mathcal{G}' \in \Pi^+$ for all pairs of affine strategies $\mathcal{G}, \mathcal{G}' \in \Pi^+$. It is easy to see disjunction closure preserves upward-closedness of strategy sets.

Proposition 2. *Disjunction closure is determinisation: $[\mathcal{P}^D]_s = [\mathcal{P}^D]_s^+ = [\mathcal{P}]_s^+$.*

Lemma 2. *For any TIOTS $\mathcal{P}$, $[\mathcal{P}^\neg]_s^+ = \{\mathcal{G}_{\mathcal{P}^\neg} \mid \forall \mathcal{G}_\mathcal{P} \in [\mathcal{P}]_s^+, h \in H : \mathcal{G}_{\mathcal{P}^\neg} \parallel_h \mathcal{G}_\mathcal{P}$ is $\bot$-free$\}$.*

Theorem 1. *Given TIOTSs $\mathcal{P}$ and $\mathcal{Q}$, $\mathcal{P} \sqsubseteq \mathcal{Q}$ iff $[\mathcal{Q}]_s^+ \subseteq [\mathcal{P}]_s^+$.*

Looking at Figure 3, the disjunction of strategies 1 and 3 produces A, while the disjunction of strategies 2 and 4 produces B. Thus $[\mathcal{P}]_s^+ = [\mathcal{Q}]_s^+$.

Relating operational composition to strategies. The operations of parallel composition, conjunction, disjunction and quotient defined on the operational models of TIOTSs (Section 2.3) can be characterised by simple operations on strategies in the game-based setting.

Lemma 3. *For $\|$-composable TIOTSs $\mathcal{P}$ and $\mathcal{Q}$, $[\mathcal{P} \parallel \mathcal{Q}]_s^+ = \{\mathcal{G}_{\mathcal{P}\|\mathcal{Q}} \mid \exists \mathcal{G}_\mathcal{P} \in [\mathcal{P}]_s^+, \mathcal{G}_\mathcal{Q} \in [\mathcal{Q}]_s^+, h \in H : \mathcal{G}_\mathcal{P} \parallel_h \mathcal{G}_\mathcal{Q} \preceq \mathcal{G}_{\mathcal{P}\|\mathcal{Q}}\}$.*

Lemma 4. *For $\vee$-composable TIOTSs $\mathcal{P}$ and $\mathcal{Q}$, $[\mathcal{P} \vee \mathcal{Q}]_s^+ = ([\mathcal{P}]_s^+ \cup [\mathcal{Q}]_s^+)^+$.*

Lemma 5. *For $\wedge$-composable TIOTSs $\mathcal{P}$ and $\mathcal{Q}$, $[\mathcal{P} \wedge \mathcal{Q}]_s^+ = [\mathcal{P}]_s^+ \cap [\mathcal{Q}]_s^+$.*

Lemma 6. *For $\%$-composable TIOTSs $\mathcal{P}$ and $\mathcal{Q}$, $[\mathcal{P}\%\mathcal{Q}]_s^+ = \{\mathcal{G}_{\mathcal{P}\%\mathcal{Q}} \mid \forall \mathcal{G}_\mathcal{Q} \in [\mathcal{Q}]_s^+, h \in H : \mathcal{G}_{\mathcal{P}\%\mathcal{Q}} \parallel_h \mathcal{G}_\mathcal{Q} \in [\mathcal{P}]_s^+\}$.*

Thus, conjunction and disjunction are the join and meet operations, and quotient produces the coarsest TIOTS s.t. $(\mathcal{P}_0\%\mathcal{P}_1)\|\mathcal{P}_1$ is a refinement of $\mathcal{P}_0$.

Theorem 2. *$\simeq$ is a congruence w.r.t. $\|$, $\vee$, $\wedge$ and $\%$ subject to composability.*

Summary. Strategy semantics has given us a weakest $\bot$-preserving congruence (i.e. $[\mathcal{P}]_s^+$) for timed specification theories based on operators for (parallel) composition, conjunction, disjunction and quotient. Strategy semantics captures nicely the game-theoretical nature as well as the operational intuition of the specification theory. In the next section, we give a more declarative characterisation of the equivalence by means of timed traces.

4 Declarative Specification Theory

In this section, we develop a compositional specification theory based on timed traces. We introduce the concept of a timed-trace structure, which is an abstract representation for a timed component. The timed-trace structure contains essential information about the component, for checking whether it can be substituted with another in a safety and liveness preserving manner.

Given any TIOTS $\mathcal{P} = \langle I, O, S, s^0, \rightarrow \rangle$, we can extract three sets of traces from $(\mathcal{P}^{\perp})^{\top}$: TP a set of timed traces leading to plain states; TE a set of timed traces leading to the error state $\perp$; and TM a set of timed traces leading to the magic state $\top$. TE and TM are extension-closed as $\top$ and $\perp$ are chaotic, while TP is prefix-closed. Due to $\top/\perp$-completion, it is easy to verify $TE \cup TP \cup TM$ gives rise to the full set of timed traces tA^*; thus TP and TE are sufficient.

However, TP and TE contain more information than necessary for our substitutive refinement, which is designed to preserve $\perp$-freeness. For instance, adding any trace $tt \in TE$ to TP should not change the semantics of the component. Based on a slight abstraction of the two sets, we can thus define a *trace structure* $\mathcal{TT}(\mathcal{P})$ as the semantics of $\mathcal{P}$.

Definition 6 (Trace structure). $\mathcal{TT}(\mathcal{P}) := (I, O, TR, TE)$, *where* $TR := TE \cup TP$ *the set of realisable traces. Obviously,* TR *is prefix-closed.*

From hereon let $\mathcal{P}_0$ and $\mathcal{P}_1$ be two TIOTSs with trace structures $\mathcal{TT}(\mathcal{P}_i) := (I_i, O_i, TR_i, TE_i)$ for $i \in \{0, 1\}$. Define $\bar{i} = 1 - i$.

The substitutive refinement relation $\sqsubseteq$ in Section 2.3 can equally be characterised by means of trace containment. Consequently, $\mathcal{TT}(\mathcal{P}_0)$ can be regarded as providing an alternative encoding of the set $[\mathcal{P}_0]_s^+$ of strategies.

Theorem 3. $\mathcal{P}_0 \sqsubseteq \mathcal{P}_1$ *iff* $TR_1 \subseteq TR_0$ *and* $TE_1 \subseteq TE_0$.

We are now ready to define the timed-trace semantics for the operators of our specification theory. Intuitively, the timed-trace semantics mimic the synchronised product of the operational definitions in Section 2.3.

Parallel composition. The idea behind parallel composition is that the projection of any trace in the composition onto the alphabet of one of the components should be a trace of that component.

Proposition 3. *If* $\mathcal{P}_0$ *and* $\mathcal{P}_1$ *are* $\|$*-composable, then* $\mathcal{TT}(\mathcal{P}_0 \parallel \mathcal{P}_1) = (I, O, TR, TE)$ *where* $I = (I_0 \cup I_1) \setminus O$, $O = O_0 \cup O_1$ *and the trace sets are given by:*

- $TE = \{tt \mid tt \restriction tA_i \in TE_i \wedge tt \restriction tA_{\bar{i}} \in TR_{\bar{i}}\} \cdot tA^*$
- $TR = TE \uplus \{tt \mid tt \restriction tA_i \in (TR_i \setminus TE_i) \wedge tt \restriction tA_{\bar{i}} \in (TR_{\bar{i}} \setminus TE_{\bar{i}})\}$

The above says tt is an error trace if the projection of tt on one component is an error trace, while the projection of tt on the other component is a realisable trace. tt is a realisable trace if tt is either an error trace or a (strictly) plain trace. tt is a (strictly) plain trace if the projections of tt on to $\mathcal{P}_0$ and $\mathcal{P}_1$ are (strictly) plain traces.

Disjunction. From any composite state in the disjunction of two components, the composition should only be willing to accept inputs that are accepted by both components, but should accept the union of outputs. After witnessing an output enabled by only one of the components, the disjunction should behave like that component. Because of the way that $\bot$ and $\top$ work in Table 1, this loosely corresponds to taking the union of the traces from the respective components.

Proposition 4. *If $\mathcal{P}_0$ and $\mathcal{P}_1$ are $\vee$-composable, then $\mathcal{TT}(\mathcal{P}_0 \vee \mathcal{P}_1) = (I, O, TR_0 \cup TR_1, TE_0 \cup TE_1)$, where $I = I_0 = I_1$ and $O = O_0 = O_1$.*

Conjunction. Similarly to disjunction, from any composite state in the conjunction of two components, the composition should only be willing to accept outputs that are accepted by both components, and should accept the union of inputs, until a stage when one of the component's input assumptions has been violated, after which it should behave like the other component. Because of the way that both $\bot$ and $\top$ work in Table 1, this essentially corresponds to taking the intersection of the traces from the respective components.

Proposition 5. *If $\mathcal{P}_0$ and $\mathcal{P}_1$ are $\wedge$-composable, then $\mathcal{TT}(\mathcal{P}_0 \wedge \mathcal{P}_1) = (I, O, TR_0 \cap TR_1, TE_0 \cap TE_1)$, where $I = I_0 = I_1$ and $O = O_0 = O_1$.*

Quotient. Quotient ensures its composition with the second component is a refinement of the first. Given the synchronised running of $\mathcal{P}_0$ and $\mathcal{P}_1$, if $\mathcal{P}_0$ is in a more refined state than $\mathcal{P}_1$, the quotient will try to rescue the refinement by taking $\top$ as its state (so that its composition with $\mathcal{P}_1$'s state will refine $\mathcal{P}_0$'s). If $\mathcal{P}_0$ is in a less or equally refined state than $\mathcal{P}_1$, the quotient will take the worst possible state without breaking the refinement.

Proposition 6. *If $\mathcal{P}_0$ dominates $\mathcal{P}_1$, then $\mathcal{TT}(\mathcal{P}_0 \% \mathcal{P}_1) = (I, O, TR, TE)$, where $I = I_0 \cup O_1$, $O = O_0 \setminus O_1$, and the trace sets satisfy:*

- $TE = TE_0 \cup \{tt \mid tt \upharpoonright tA_1 \notin TR_1\} \cdot tA^*$
- $TR = TE \uplus \{tt \mid tt \in (TR_0 \setminus TE_0) \wedge tt \upharpoonright tA_1 \in (TR_1 \setminus TE_1)\}$.

The above says tt is an error trace if either tt is an error trace in $\mathcal{P}_0$ or the projection of tt on $\mathcal{P}_1$ is not a realisable trace. A strictly plain trace must have strictly plain projections onto $\mathcal{P}_0$ and $\mathcal{P}_1$.

Mirroring of trace structures is equally straightforward: $\mathcal{TT}(\mathcal{P}_0)^\neg = (O_0, I_0, tA^* \setminus TE_0, tA^* \setminus TR_0)$. Consequently, quotient can also be defined as the derived operator $(\mathcal{TT}(\mathcal{P}_0)^\neg \parallel \mathcal{TT}(\mathcal{P}_1))^\neg$.

5 Comparison with Related Works

Our framework can be seen as a linear-time alternative to the timed specification theories of [2] and [3], albeit with significant differences. The specification theory in [3] also introduces parallel, conjunction and quotient, but uses timed alternating simulation as refinement, which does not admit the weakest precongruence.

An advantage of [3] is the algorithmic efficiency of branching-time simulation checking and the implementation reported in [10].

The work of [2] on timed games also bears conceptual similarities, although they do not define conjunction and quotient. We adopt most of the game rules in [2], except that, due to our requirement that proposed delay moves are maximal delays allowed by a strategy, a play cannot have consecutive delay moves. This enables us to avoid the complexity of time-blocking strategies and blame assignment, but does not ensure non-Zenoness[8]. Secondly, we do not use timestop/semi-timestop to model time errors (i.e. bounded-liveness errors). Rather, we introduce the explicit inconsistent state $\perp$ to model both time and immediate (i.e. safety) errors. This enables us to avoid the complexity of having two transition relations and well-formedness of timed interfaces.

Based on linear time, our timed theory owes much to the pioneering work of trace theories in asynchronous circuit verification, such as Dill's trace theory [9]. Our mirror operator is essentially a timed extension of the mirror operator from asynchronous circuit verification [15]. The definition of quotient based on mirroring (for the untimed case) was first presented by Verhoeff as his Factorisation Theorem [14].

In comparison with our untimed theory [1], our timed extension requires new techniques (e.g. those related to timestop) to handle delay transitions since time can be modelled neither as input nor as output. In the timed theory, the set of realisable traces (TR) is not required to be input-enabled, which is necessary for the set of untimed traces in [1]. Thus, the domain of trace structures is significantly enlarged. Furthermore, the timed theory supports the modelling of liveness assumptions/guarantees, with the checking of such violations reducing to $\perp$-reachability. Therefore, finite traces suffice to model and verify liveness properties, whereas in contrast, the untimed theory must employ infinite traces to treat liveness in a proper way.

We briefly mention other related works, which include timed modal transition systems [11, 12], the timed I/O model [5, 13] and embedded systems [18, 19].

6 Conclusions

We have formulated a rich compositional specification theory for components with real-time constraints, based on a linear-time notion of substitutive refinement. The operators of hiding and renaming can also be defined, based on our previous work [8]. We believe that our theory can be reformulated as a timed extension of Dill's trace theory [9]. Future work will include an investigation of realisability and assume-guarantee reasoning.

Acknowledgments. The authors are supported by EU FP7 project CONNECT, ERC Advanced Grant VERIWARE and EPSRC project EP/F001096.

[8] Zeno behaviours (infinite action moves within finite time) in a play are not regarded as abnormal behaviours in our semantics.

References

1. Chen, T., Chilton, C., Jonsson, B., Kwiatkowska, M.: A compositional specification theory for component behaviours. In: ESOP'12. Volume 7211 of LNCS., Springer-Verlag (2012) 148–168
2. de Alfaro, L., Henzinger, T.A., Stoelinga, M.: Timed interfaces. In: EMSOFT'02. Volume 2491 of LNCS. Springer-Verlag (2002) 108–122
3. David, A., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: Timed I/O automata: a complete specification theory for real-time systems. In: HSCC '10, ACM (2010) 91–100
4. de Alfaro, L., Henzinger, T.A.: Interface automata. SIGSOFT Softw. Eng. Notes **26** (2001) 109–120
5. Kaynar, D.K., Lynch, N.A., Segala, R., Vaandrager, F.W.: Timed I/O Automata: A mathematical framework for modeling and analyzing real-time systems. In: RTSS. (2003)
6. Lim, W.: Design methodology for stoppable clock systems. Computers and Digital Techniques, IEE Proceedings E **133** (1986) 65 –72
7. Moore, S., Taylor, G., Cunningham, P., Mullins, R., Robinson, P.: Using stoppable clocks to safely interface asynchronous and synchronous subsystems. In: AINT (Asynchronous INTerfaces) Workshop, Delft, Netherlands (2000)
8. Wang, X., Kwiatkowska, M.Z.: On process-algebraic verification of asynchronous circuits. Fundam. Inform. **80** (2007) 283–310
9. Dill, D.L.: Trace theory for automatic hierarchical verification of speed-independent circuits. ACM distinguished dissertations. MIT Press (1989)
10. David, A., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: Ecdar: An environment for compositional design and analysis of real time systems. In: ATVA. Volume 6252 of LNCS., Springer (2010) 365–370
11. Bertrand, N., Legay, A., Pinchinat, S., Raclet, J.B.: A compositional approach on modal specifications for timed systems. In: ICFEM. Volume 5885 of LNCS., Springer (2009) 679–697
12. Cerans, K., Godskesen, J.C., Larsen, K.G.: Timed modal specification - theory and tools. In: CAV. (1993) 253–267
13. Berendsen, J., Vaandrager, F.W.: Compositional abstraction in real-time model checking. In: FORMATS. Volume 5215 of LNCS., Springer (2008) 233–249
14. Verhoeff, T.: A Theory of Delay-Insensitive Systems. PhD thesis, Dept. of Math. and C.S., Eindhoven Univ. of Technology (1994)
15. Zhou, B., Yoneda, T., Myers, C.: Framework of timed trace theoretic verification revisited. IEICE Trans. on Information and Systems **85** (2002) 1595–1604
16. Chilton, C., Kwiatkowska, M., Wang, X.: Revisiting timed specification theories: A linear-time perspective. Technical Report RR-12-04, Department of Computer Science, University of Oxford (2012)
17. Wang, X.: Maximal Confluent Processes. In: Petri Nets'12. Volume 7347 of LNCS., Springer-Verlag (2012)
18. Thiele, L., Wandeler, E., Stoimenov, N.: Real-time interfaces for composing real-time systems. In: EMSOFT. (2006)
19. Lee, I., Leung, J., Song, S.: Handbook of Real-Time and Embedded Systems. Chapman (2007)

A.3 Automatic Synthesis of Modular Connectors via Composition of Protocol Mediation Patterns

Automatic Synthesis of Modular Connectors via Composition of Protocol Mediation Patterns

Paola Inverardi and Massimo Tivoli
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica
Università degli Studi dell'Aquila, Italy
Email: {paola.inverardi,massimo.tivoli}@univaq.it

***Abstract*—Ubiquitous and pervasive computing promotes the creation of an environment where Networked Systems (NSs) eternally provide connectivity and services without requiring explicit awareness of the underlying communications and computing technologies. In this context, achieving interoperability among heterogeneous NSs represents an important issue. In order to mediate the NSs interaction protocol and solve possible mismatches, connectors are often built. However, connector development is a never-ending and error-prone task and prevents the eternality of NSs. For this reason, in the literature, many approaches propose the automatic synthesis of connectors. However, solving the connector synthesis problem in general is hard and, when possible, it results in a monolithic connector hence preventing its evolution. In this paper, we define a method for the automatic synthesis of modular connectors, each of them expressed as the composition of independent mediators. A modular connector, as synthesized by our method, supports connector evolution and performs correct mediation.**

I. Introduction

The near future envisions an ubiquitous and pervasive computing environment that enables heterogeneous Networked Systems (NSs) to provide and access software services without requiring an explicit awareness of the underlying communications and computing technologies [1]. In such an environment, a problem that arises when integrating and composing different NSs is related to the problem of achieving interoperability among heterogeneous NSs by solving possible protocol mismatches. They can occur while the NSs interact with each other to accomplish some common task.

A widely used technique to cope with this problem is to build connectors [2], [3] that bridge the communication among heterogenous protocols and coordinate their interaction. However, due to the potentially infinite number of different available protocols, connector development is a never-ending and error-prone task and prevents the eternality of NSs. Nowadays, the efficacy of integrating and composing NSs is proportional to the level of interoperability of the systems' respective underlying technologies. For this reason, in the literature, starting from the pioneering work in [4], many approaches propose the automatic synthesis of connectors, see [5]–[8] just to cite a few.

As a matter of fact, in general, the connector synthesis problem is hard in the sense that not all possible protocol mismatches are solvable. For instance, building a connector that reconciles the component interaction by reordering certain sequences of exchanged messages can lead to unbounded executions. As shown in [9], a suitable termination criterion can be defined with the aim of under-approximating unbounded interactions by means of bounded ones whenever a pattern of behaviour indicating potential infinity occurs. Thus, practical solutions can only deal with a combination of specific mediation patterns that correspond to tractable protocol mismatches [10]–[13]. However, these solutions eventually result in a monolithic connector hence preventing evolution, and making synthesis and maintenance of the connector code a difficult task.

In this paper, we define a method for the automatic synthesis of modular connectors. A modular connector is represented as a suitable composition of independent *mediators*. Each mediator can be seen as a basic (sub-)connector that realizes a specific mediation pattern, which corresponds to the solution of a recurring protocol mismatch. The advantage of our connector decomposition is twofold: (i) it is *correct*, i.e., as for its monolithic version, the mediation logic performed by a modular connector is free from possible mismatches; and (ii) it promotes connector *evolution*, hence also easing the synthesis and maintenance of its implementation code. To show (i), we formally define the semantics of protocols (as well as of mediators and connectors) by using a revised version of the *Interface Automata* (IA) theory described in [14]. Then, we prove that a modular connector for two protocols P and R enjoys the same correctness properties of the monolithic connector obtained by expressing the synthesis problem as a

This work has been partially supported by the FET project CONNECT No 231167.

quotient problem between P and R [15]. Concerning the set of considered mediation patterns and, hence, connector modularization, our synthesis method relies on a revised version of the connector algebra described in [16]. It is an algebra for reasoning about protocol mismatches where basic mismatches can be solved by suitably defined primitives, while complex mismatches can be settled by composition operators that build connectors out of simpler ones. We revise the original algebra by adding an iterator operator and by giving its semantics in terms of our revised IA theory. For (ii), we make use of a case study in the e-commerce domain to illustrate that relevant changes can be applied on a modular connector by simply acting on its constituent mediators, without entirely re-synthesizing its protocol.

The paper is organized as follows. Section II puts the bases for the definition of our synthesis method by discussing our revised version of both the IA theory in [14] and the connector algebra in [16]. Section III introduces the *purchase order mediation scenario* that we use as case study in the sequel of the paper. In Section IV, we formalize our synthesis method and illustrate it at work on the case study. In Section V, we state correctness of our method and, by means of the case study, we show how it supports connector evolution. Section VI discusses related work, and Section VII provides final remarks and future research directions.

II. Preamble on the Synthesis Method

At connector synthesis stage, we assume that a NS comes together with a IA-based specification of its interaction protocol. The interaction protocol of a NS expresses the order in which *input* and *output* actions are performed while the NS interacts with environment. In our setting, actions are used to abstract messages that can be sent (outputs) or received (inputs) by a NS. Inputs are received from and controlled by the environment, whereas outputs are controlled and emitted by the NS. A NS can perform also *hidden* actions corresponding to internal computation.

In this section, we instantiate some definitions from the IA theory in [14] to our context and, when needed for the purposes of automated connector synthesis, we also add new ones.

Definition 1 (*Interaction Protocol Specification*)
An Interaction Protocol Specification *(IPS) P is a tuple $(A_P^I, A_P^O, A_P^H, S_P, s_P^0, \delta_P)$, where A_P^I, A_P^O, A_P^H are disjoint sets referred to as* input, output, *and* hidden *actions (the union of which we denote by A_P), S_P is a finite set of* states *with $s_P^0 \in S_P$ being the designated* initial state, *and $\delta_P : S_P \times A_P \to S_P$ is the partial* transition function.

Intuitively, from a state, the NS may either emit any output that is enabled according to its IPS or perform internal computation. If the environment supplies an input that is enabled, the reaction of the NS is according to its IPS. If the input is not enabled, this causes an *inconsistency*.

Let a be an action, we denote with $\overline{a}$ its *complement*. If a is an input action then $\overline{a}$ is the corresponding output action, and vice versa. When a is a hidden action, $\overline{a}$ is hidden as well and its label is the one of a followed by ';'. Abusing notation, we extend the complement also to IPSs. That is, let P be an IPS, then $\overline{P}$ denotes its complement and it is P where all input, output, and hidden actions have been complemented. Furthermore, we consider a special kind of IPS denoted by I and called *identity*. It is defined as the IPS $(\emptyset, \emptyset, \emptyset, \{s_I^0\}, s_I^0, \emptyset)$. To give the possibility to express IPSs that take a message as input and forward the same message as output, given an action a, we consider also the action a' as semantically equivalent to a (yet syntactically different). We write $s \xrightarrow{a}_P s'$ to denote that $\delta_P(s,a) = s'$ (or, equivalently, that $(s,a,s') \in \delta_P$). An action a is *enabled* in s, if $\delta_P(s,a)$ is defined. $A_P(s)$ denotes the set of actions in P that are enabled in s. We denote with $s \stackrel{a}{\Longrightarrow}_P s'$ a sequence of internal actions starting from s, terminating to s', and with an observable action a in some point in the middle of the sequence. We write $s \stackrel{a}{\rightsquigarrow}_P$ to denote that, from s, P can perform a sequence of hidden actions terminating with the observable action a. Abusing notation, $\rightsquigarrow_P(s,a)$ denotes the set of states, in P, that are reachable from s by performing a sequence of hidden actions terminating with the observable action a.

Definition 2 (*Traces of an IPS*)
Let $P = (A_P^I, A_P^O, A_P^H, S_P, s_P^0, \delta_P)$ be an IPS, a trace *of P is a $t_P \in ((A_P^I \cup A_P^O)^* \cup \{\epsilon\})$ defined in such a way that $t_P = \epsilon \vee \exists n > 0, s_0^P, \ldots, s_n^P \in S_P : t_P = a_1 a_2 \ldots a_n \wedge s_0^P \stackrel{a_1}{\Longrightarrow} \ldots \stackrel{a_n}{\Longrightarrow} s_n^P$, where ϵ denotes the so called* empty trace.

We denote with A_{t_P} the *set of actions* in t_P, and with $|t_P|$ the length of t_P. Furthermore, we denote with $t_P(a)$ the first position of the action a in t_P. Finally, we denote with $Tr(P)$ the *set of traces* of P.

From hereon let $P = (A_P^I, A_P^O, A_P^H, S_P, s_P^0, \delta_P)$ and $R{=}(A_R^I, A_R^O, A_R^H, S_R, s_R^0, \delta_R)$ be two IPSs. P and R may only be composed if their action sets are compatible with each other. IPSs P and R are *composable* if $A_P^H \cap A_R = \emptyset$, $A_P \cap A_R^H = \emptyset$, $A_P^I \cap A_R^I = \emptyset$, and $A_P^O \cap A_R^O = \emptyset$. We denote with $common(P,R)$ the set $A_P \cap A_R$ of common actions. Note that if P and R are composable then $common(P,R) = (A_P^I \cap A_R^O) \cup (A_P^O \cap A_R^I)$. To define the parallel composition of composable IPSs, we use a *product* operation that accounts for possible semantically equivalent actions.

Definition 3 (*Product of two IPSs*)
The product of P and R is an IPS $P \otimes R = (A_{P\otimes R}^I, A_{P\otimes R}^O, A_{P\otimes R}^H, S_P \times S_R, (s_P^0, s_R^0), \delta_{P\otimes R})$, where:

- $A_{P\otimes R}^I = (A_P^I \cup A_R^I) \setminus common(P,R)$;
- $A_{P\otimes R}^O = (A_P^O \cup A_R^O) \setminus common(P,R)$;
- $A_{P\otimes R}^H = A_P^H \cap A_R^H \cap common(P,R)$;
- $(p,r) \xrightarrow{a}_{P\otimes R} (p',r') \Leftrightarrow$
  - $p \xrightarrow{a\ (resp.,\ a')}_P p' \wedge \delta_R(r,a')$ *(resp.,* $\delta_R(r,a)$*) is not defined* $\wedge\ r = r' \wedge a$ *(resp.,* a'*)* $\notin common(P,R)$;
  - $p = p' \wedge \delta_P(p,a')$ *(resp.,* $\delta_P(p,a)$*) is not defined* $\wedge$ $r \xrightarrow{a\ (resp.,\ a')}_R r' \wedge a$ *(resp.,* a'*)* $\notin common(P,R)$;

– $p \xrightarrow{a\ (resp.,\ a')}_P p' \wedge r \xrightarrow{a\ (resp.,\ a')}_R r' \wedge$
$a\ (resp.,\ a') \in common(P,R)$;
– $p \xrightarrow{a\ (resp.,\ a')}_P p' \wedge r \xrightarrow{a'\ (resp.,\ a)}_R r'$.

Unfortunately, the product can introduce a number of inconsistencies when one of the two protocols is willing to offer an output action in the common alphabet, but the second is not able to offer, possibly after a sequence of hidden actions, the corresponding input action (accounting also for possible semantically equivalent actions). We denote with $Inconsistencies(P,R)$ the set of states in the product $P \otimes R$ from which inconsistencies can arise. The kernel of the inconsistencies in $P \otimes R$, with P and R composable, is the set of states (p,r) for which: either (i) there is some $a \in common(P,R)$ (resp., $a' \in common(P,R)$) such that one of p and r can make an a-labelled (resp., a'-labelled) output transition, but the other cannot match it with the corresponding input transition; or (ii) one of p and r can make an a-labelled (resp., a'-labelled) output transition, but the other cannot match it with the semantically equivalent input transition. $Inconsistencies(P,R)$ is then the set of those states in the kernel, plus those that can reach a state in the kernel by a sequence of transitions labelled by either output or hidden actions.

Definition 4 (*Composition of two IPSs*)
The composition *of two composable IPSs P and R, written as $P||R$, is defined to be $P \otimes R$ after pruning all states in $Inconsistencies(P,R)$, providing the initial state (s_0^P, s_0^R) is contained within the remaining automaton. Otherwise, the composition is undefined.*

As formally proven in [14], $||$ is a *compositional* operator meaning that, given three composable IPSs P, Q, and R, then $(P||Q)||R = P||(Q||R)$.

As it will be clear in Section V, to state correctness of our synthesis method, we use a notion of *refinement* based on a version of *alternating simulation* [17] that accounts for both hidden actions and semantically equivalent ones. Informally, R refines P if all input steps of P can be simulated by R, and all the output steps of R can be simulated by P, considering that internal steps of P and R are independent and an observable step can be simulated by a semantically equivalent one. We make use of a "semantic" inclusion operator, denoted by $\subseteq_{sem}$, between sets of actions. Its meaning is the same as $\subseteq$, but it accounts also for semantically equivalent actions. That is, given two sets of actions S and S', if $S \subseteq S'$ then $S \subseteq_{sem} S'$ and, given an action a, either $(\{a\} \cup S) \subseteq_{sem} (\{a'\} \cup S')$ or $(\{a'\} \cup S) \subseteq_{sem} (\{a\} \cup S')$.

Definition 5 (*Refinement between IPSs*)
R refines *P, denoted by $P \succeq R$ if the following conditions hold:*

- $A_P^I \subseteq_{sem} A_R^I \wedge A_R^O \subseteq_{sem} A_P^O$;
- *there exists an* alternating simulation *as a binary relation $\succeq \subseteq S_P \times S_R$ such that for all states $s \in S_P$ and $r \in S_R$, with $s \succeq r$, the following conditions hold:*
  - (ii.1) $\{x | s \overset{x}{\rightsquigarrow}_P \wedge x \in A_P^I\} \subseteq_{sem} \{y | r \overset{y}{\rightsquigarrow}_R \wedge y \in A_R^I\}$;
  - (ii.2) $\{y | r \overset{y}{\rightsquigarrow}_R \wedge y \in A_R^O\} \subseteq_{sem} \{x | s \overset{x}{\rightsquigarrow}_P \wedge x \in A_P^O\}$;
  - (ii.3) $\forall z \in \{x | s \overset{x}{\rightsquigarrow}_P \wedge x \in A_P^I\} \cup \{y | r \overset{y}{\rightsquigarrow}_R \wedge y \in A_R^O\}$, $r' \in \rightsquigarrow_R(r,z) : \exists s' \in \rightsquigarrow_P(s,z) : s' \succeq r'$;
- $s_P^0 \succeq s_R^0$.

Refinement between IPSs is a *preorder* (i.e., reflexive and transitive). Note that P and $\overline{P}$ are always composable and, under refinement, $P||\overline{P}$ and I are equivalent, i.e., $P||\overline{P} \succeq I$ and $I \succeq P||\overline{P}$. The same holds for $P||I$ and P, i.e., $P \succeq P||I$ and $P||I \succeq P$. Furthermore, let P' and R' be two IPSs, refinement is *compositional* meaning that $P||P' \succeq R||R'$, if both $P \succeq R$ and $P' \succeq R'$.

As already mentioned in Section I, in order to reason about protocol mismatches, we consider a revised version of the connector algebra described in [16]. In the following, we report only the portion of the algebra that is relevant for the purposes of this work.

From hereon let $\mathcal{A}$ be the universal set of actions. The primitives of the connector algebra $\mathcal{AP}(\mathcal{A})$, corresponding to specific protocol mismatches, are described below.

1) **Extra send:** it concerns the possibility for a NS to generate either a redundant or an additional message a. Such a mismatch can be solved by means of a mediator that consumes a. It is modeled by the primitive $Cons(a)$ that takes a as input.
2) **Missing send:** it occurs when a NS expects either a redundant message or a message a that is not sent by another NS. It can be solved by introducing a mediator that generates a. It is modeled by the primitive $Prod(a)$ that produces a as output.
3) **Signature mismatch:** two messages a and b of two different NSs can be functionally compatible yet syntactically inconsistent. A mediator that performs the needed translation can solve this mismatch, i.e., $Trans(a,b)$ that takes a as input and produces b as output.
4) **Split message mismatch:** a NS may expect to receive a message a as a sequence of fragments of a. If message a can be decomposed into $a_1, \ldots, a_n$, then the mismatch can be solved by $Split(a, [a_1, \ldots, a_n])$ which takes a as input and produces an ordered sequence $a_1, \ldots, a_n$ as output.
5) **Merge message mismatch:** it is symmetric to the previous one. The mismatch can be solved by the primitive $Merge([a_1, \ldots, a_n], a)$ which takes an ordered sequence $a_1, \ldots, a_n$ as input, and produces a as output.
6) **Reordering mismatch:** a NS expects to receive messages in an order different from the order used by the sending NS. It can be solved by a reordering primitive $Order([a_1, \ldots, a_n], \pi, [a'_1, \ldots, a'_n])$, where π is a permutation of $\{1, \ldots, n\}$. The primitive takes as input the ordered sequence $a_1, \ldots, a_n$ from one NS , and produces the ordered sequence $a'_{\pi(1)}, \ldots, a'_{\pi(n)}$ as output for another NS.

The syntax of a term t in $\mathcal{AP}(\mathcal{A})$ is given by:
$t ::= t \odot t \mid t^* \mid (t) \mid p$

$p ::= \mid Cons(a) \mid Prod(a) \mid Trans(a,b) \mid$
$Split(a,[a_1,\ldots,a_n]) \mid Merge([a_1,\ldots,a_n],a) \mid$
$Order([a_1,\ldots,a_n],\pi,[a'_1,\ldots,a'_n])$

where $a, a_i, a'_i, b \in \mathcal{A}$ and π is a permutation of $\{1,\ldots,n\}$. The symbol $\odot$ is a binary operator called *plugging*, and * is an unary operator called *iterator*.

The semantics of $\mathcal{AP}(\mathcal{A})$ is given in terms of a function $[\![\cdot]\!] : \mathcal{AP}(A) \rightarrow \mathcal{IPS} \cup \{Err\}$, where $\mathcal{IPS}$ is the universal set of IPSs and Err represents the undefined IPS, i.e., it has no states. For any term t, the denotation $[\![t]\!]$ is defined inductively.

If t is a primitive, then $[\![t]\!]$ is the corresponding IPS, providing the parameters are well-defined (otherwise $[\![t]\!] = Err$).

If t is a compound term, such as $t = p \odot r$, then $[\![t]\!]$ is given by: if either $[\![p]\!]$ or $[\![r]\!]$ is equal to Err, then $[\![t]\!] = Err$. Alternatively, if $[\![p]\!]$ and $[\![r]\!]$ are not composable or $[\![p]\!] \mid\mid [\![r]\!]$ is not defined, then $[\![t]\!] = Err$. Otherwise, $[\![t]\!] = [\![p]\!] \mid\mid [\![r]\!]$.

If t is an iterative term, such as $t = p^*$, then $[\![t]\!]$ is given by the IPS $[\![p]\!]$ suitably adjusted, by means of (hidden) ϵ-transitions, for allowing cyclic behaviour. This can be easily achieved by exploiting the well-known Thompson's algorithm to produce finite state automata from regular expressions [18]. Otherwise, if $[\![p]\!]$ is undefined, then $[\![t]\!] = Err$.

III. The Purchase Order Mediation Scenario

We borrow a case study from [19], slightly changed to better show the peculiarities of our method. The case study concerns the so called *Purchase Order Mediation scenario* from the Semantic Web Service (SWS) Challenge[1]. As stated in [19], "*it represents a typical real-world problem that is as close to industrial reality as practical... This scenario highlights the various mismatches that can be encountered when making heterogeneous systems interoperable*". The scenario considers two NSs implemented using different standards and protocols: the *Blue Service* (BS) and the *Moon Client* (MC).

Figures 1 and 2 show the IPSs of BS and MC, respectively. The box that encloses the transition system corresponds to the protocol interface. Labelled arrows pointing at the interface correspond to inputs, whereas arrows departing from the interface correspond to outputs. Both BS and MC do not perform hidden actions.

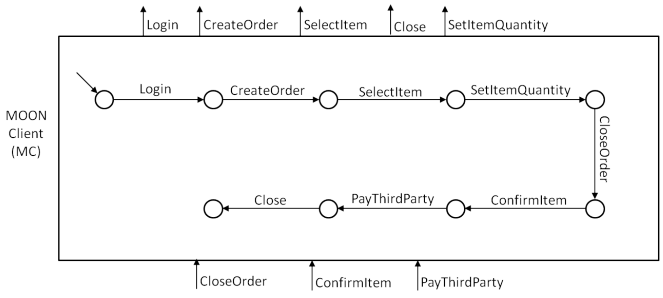


Fig. 1. MC: Moon Client Interaction Protocol

[1] http://sws-challenge.org/wiki/.

MC orders products by assuming to interact with a Moon Service according to the following protocol: MC has to perform authentication (Login) to prove that it represents a known and authorized customer; if authentication succeeds, an order can be created (CreateOrder) by starting from an empty cart; then, individual items can be added to the created order. Thus, an item is selected (SelectItem) by also specifying the needed quantity (SetItemQuantity). At this stage, MC is asked, from the Moon Service, to confirm the addition of the item to the order (CloseOrder followed by ConfirmItem). Finally, the order payment is asked by a third-party payment system (PayThirdParty) so as the order can be closed (Close).

BS is a service providing purchase order functionalities. A client can initiate a purchase by starting an order (StartOrder) and adding items to it (AddItemtoOrder). For each item belonging to the order, BS asks its clients to confirm the item addition (GetConfirmation) and, then, to place the order by means of the client identifier (PlaceOrder). Finally, a client can quit the session (Quit).

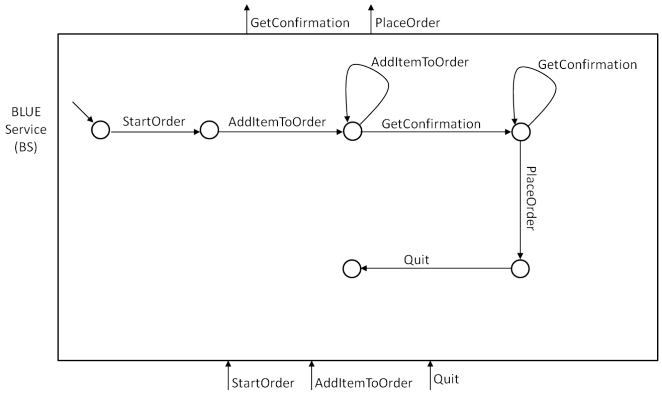


Fig. 2. BS: Blue Service Interaction Protocol

MC cannot communicate with BS due to the following protocol mismatches, of two different types.

Communication mismatches concern the semantics and granularity of the protocol actions. For instance, a client of BS provides its identifier while placing the order, whereas MC has to authenticate before performing any operation. Furthermore, BS provides a single operation to add an item, with the needed quantity, to the order, whereas MC expects to use two different operations, one for the addition and one for the quantity specification. To solve these kind of mismatches it is necessary to assume and use ontology knowledge in order to align the two protocols to the same concepts and language.

Coordination mismatches concern the control structure of the protocols and can be solved by means of the mediator that can mediate the conversation between the two protocols so that they can actually interact. For instance, BS requires its clients to confirm the ordered items and then place the order, whereas MC expects to confirm the ordered items only once the order is placed. Finally, BS allows the addition of several

kinds of items in the same order, whereas MC performs the addition of only one kind of item per order.

As stated in [19], MC and BS "*are provided by the SWS-Challenge organizers and can not be altered (although their description may be semantically enriched)*". In particular, in [19], by exploiting an ontology [20] in the domain of *purchase processes*, the description of both MC and BS has been semantically enriched. This domain ontology is denoted by DO and shown in Figure 3.

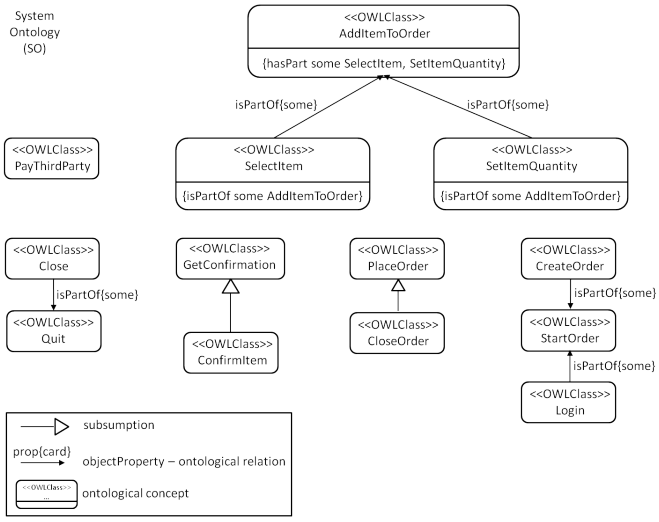


Fig. 3. DO: Domain Ontology

DO shows the relations holding between the various concepts used by MC and BS as purchase order systems. Typically, ontologies account for two fundamental relations between concepts: *subsumption* and *aggregation* [21]. A concept a is *subsumed by* a concept b, in a given ontology O, if in every model of O the set denoted by a is a subset of the set denoted by b. A concept a is *an aggregate of* concepts $b_1, \ldots, b_n$ if the latter are part of the former. It is worth to mention that our use of the ontology concept is specific of the CONNECT project (this work is part of). Thus, in the following, we will exploit these notions to our purposes. That is, concepts in DO correspond to NS input/output actions. The two relations between concepts are, then, used to account for the granularity of the data that define the structure of the messages exchanged by the respective input/output actions. For example, the message associated to the request of AddItemToOrder is an aggregate of the messages associated to the requests of SelectItem and SetItemQuantity.

IV. Synthesis of Modular Connectors

A mediator has an input-output behaviour (not necessarily strictly sequential, e.g., for allowing reordering of messages), and it is a "reactive" software entity harmonizing the interaction between heterogeneous NSs by intercepting output messages from one NS and eventually issuing to another NS the *co-related* input messages. Message co-relations can be inferred by taking into account ontological information, which is characterized as follows. We recall that the following definitions apply in the scope of our problem that is defined by two NSs enriched with the respective IPS and ontologies as defined in the previous section.

Definition 6 (*Protocol Ontology*)
A protocol ontology O *is a triple* (C_O, S_O, A_O)*, where* C_O *is the set of* ontological concepts *and it is partitioned into the two disjoint sets* IC_O *and* OC_O *of* input *and* output *ontological concepts, respectively.* S_O *and* A_O *are the* subsumption *and* aggregation *relations among concepts, respectively. The following properties hold:*

- $(C_O \subseteq \mathcal{A}) \wedge (C_O = IC_O \cup OC_O) \wedge (IC_O \cap OC_O = \emptyset)$;
- $S_O \subseteq IC_O \times OC_O$;
- $A_O \subseteq IC_O \times OC_O^n \wedge n > 0$.

If $(a, b) \in S_O$ then we write that a *is subsumed by* b, and (a, b) is called *subsumption pair*. This means that the output action b from a component C must precede the input action a from a (different) component C'. In other words, since the set of data constituting a is a subset of those constituting b, to build the message associated to a, one needs to process the data contained in the message associated to b first. A mediator that makes C and C' able to interoperate would take b as input from C and send, as co-related message, a as output to C'. If $(a, b_1, \ldots, b_n) \in A_O$ then we write that $b_1, \ldots, b_n$ *are part of* a. Furthermore, $(a, b_1, \ldots, b_n)$ is called *aggregation tuple*. If $b_1, \ldots, b_n$ are part of a then the latter can be built by merging the messages associated to $b_1, \ldots, b_n$. Thus, a mediator would take $b_1, \ldots, b_n$ as input in any order, and send a as the merge of $b_1, \ldots, b_n$, plus possible additional data explicitly specified in O for a.

We write that a protocol ontology $O{=}(IC_O{\cup}OC_O,S_O,A_O)$ is *valid* for protocols P and R if and only if $IC_O = A_P^I \cup A_R^I$ and $OC_O = A_P^O \cup A_R^O$.

Coming back to our case study, let $O{=}(IC_O{\cup}OC_O,S_O,A_O)$ be the protocol ontology shown in Figure 3 then:

$IC_O{=}\{ConfirmItem,PayThirdParty,CloseOrder,$
$\quad StartOrder,AddItemToOrder,Quit\}$;
$OC_O{=}\{GetConfirmation,Login,CreateOrder,SelectItem,$
$\quad SetItemQuantity,PlaceOrder,Close\}$;
$S_O{=}\{(ConfirmItem,GetConfirmation),$
$\quad (CloseOrder,PlaceOrder)\}$;
$A_O{=}\{(AddItemToOrder,SelectItem,SetItemQuantity),$
$\quad (Quit,Close),$
$\quad (StartOrder,Login,CreateOrder)\}$;

and it is a valid protocol ontology for MC and BS.

We, now, formalize our synthesis method as organized into two phases described in Sections IV-A and IV-B, respectively. They are performed only if the protocol ontology is valid.

A. Automatic Synthesis of Communication Mediators

In this section by defining the semantics of subsumption pairs and aggregation tuples in terms of IPSs we synthesize primitive mediators. We do this by exploiting the primitives

and operators of $\mathcal{AP}(\mathcal{A})$, hence following a modular approach. We call them *communication mediators* because they aim at solving communication mismatches among heterogeneous interaction protocols by *aligning* their different alphabets according to the defined ontological relations (Figure 3). For instance, the communication mediators for the case study introduced in Section III are:

M_1=$[\![Split(GetConfirmation, [ConfirmItem, x_1]) \odot Cons(x_1)]\!]$;
M_2=$[\![Split(PlaceOrder, [CloseOrder, x_2]) \odot Cons(x_2)]\!]$;
M_3=$[\![Trans(SelectItem, x_3) \odot Trans(SetItemQuantity, x_4) \odot Prod(x_5) \odot Merge([x_3, x_4, x_5], AddItemToOrder)]\!]$;
M_4=$[\![Trans(Close, x_6) \odot Prod(x_7) \odot Merge([x_6, x_7], Quit)]\!]$;
M_5=$[\![Trans(Login, x_8) \odot Trans(CreateOrder, x_9) \odot Prod(x_{10}) \odot Merge([x_8, x_9, x_{10}], StartOrder)]\!]$.

Definition 7 (*Semantics of Subsumption Pairs*)
Let $O = (C_O, S_O, A_O)$ be a protocol ontology, the semantics *of $(a, b) \in S_O$ is given by $[\![Split(b, [a, x]) \odot Cons(x)]\!]$.*

Definition 8 (*Semantics of Aggregation Tuples*)
Let $O = (C_O, S_O, A_O)$ be a protocol ontology, the semantics *of $(a, b_1, \ldots, b_n)$ is given by $[\![Trans(b_1, x_1) \odot \ldots \odot Trans(b_n, x_n) \odot Prod(x_{n+1}) \odot Merge([x_1, \ldots, x_n, x_{n+1}], a)]\!]$.*

For the purposes of *alphabet alignment*, when synthesized out of a subsumption pair (a, b), a mediator is used as a *wrapper for the output action b of a protocol*, which is taken as input by the mediator. Instead, when synthesized out of an aggregation tuple $(a, b_1, \ldots, b_n)$, a mediator is used as a *wrapper for the input action a of a protocol*, which is sent as output by the mediator. Thus, we define a further derived composition operator called *wrapping*. P can be *wrapped* by R on action a if and only if *P and R are composable*, $a \in Common(P, R)$, and R is a communication mediator.

Definition 9 (*Wrapping of an IPS*)
The wrapping of P by a communication mediator R, on action a, is an IPS $P \triangleleft_a R = (A^I_{P\otimes R}, A^O_{P\otimes R}, A^H_{P\otimes R}, S_{P\triangleleft_a R}, s^0_P, \delta_{P\triangleleft_a R})$, where:

- $S_{P\triangleleft_a R} = (S_P \cup S_R)$;
- $\delta_{P\triangleleft_a R} = (\delta_P \cup \delta_R) \setminus \{(s, a, s') | s, s' \in (S_P \cup S_R)\} \cup \{(p, \epsilon_{s^0_R}; , s^0_R) | (p, a, p') \in \delta_P \wedge \delta_R(S^0_R, a)$ *is not defined*$\} \cup \{(r, \epsilon_{p'}; , p') | (p, a, p') \in \delta_P \wedge r \neq s^0_R \wedge \delta_R(r, a)$ *is defined*$\} \cup \{(p, \epsilon_{r'}; , r') | (p, a, p') \in \delta_P \wedge \delta_R(s^0_R, a)$ *is defined*$\}$.

Definition 10 (*Alphabet Alignment*)
Given P and R to be mediated, let $O = (C_O, S_O, A_O)$ be the valid protocol ontology specification for P and R. Let $S_1, \ldots, S_n$ be the communication mediators synthesized according to S_O, and let $A_1, \ldots, A_h$ be those synthesized according to A_O. The algorithm for the alphabet alignment *step is as follows:*

procedure *Alignment*
input: P, R, O, $S_1,\ldots,S_n$, $A_1,\ldots,A_h$
output: $AlignedProtocols$
1: $AlignedProtocols := \emptyset$
2: ***for* each** $j : (A^I_P \cap A^O_{A_j} \neq \emptyset) \wedge (A^O_{A_j} = \{a\})$ ***do***
3: $\quad P := P \triangleleft_a A_j$
4: ***end for***
5: ***for* each** $k : (A^O_P \cap A^I_{S_k} \neq \emptyset) \wedge (A^I_{S_k} = \{b\})$ ***do***
6: $\quad P := P \triangleleft_a S_k$
7: ***end for***
8: $AlignedProtocols := AlignedProtocols \cup \{P\}$
9: ***for* each** $j : (A^I_R \cap A^O_{A_j} \neq \emptyset) \wedge (A^O_{A_j} = \{a\})$ ***do***
10: $\quad R := R \triangleleft_a A_j$
11: ***end for***
12: ***for* each** $k : (A^O_R \cap A^I_{S_k} \neq \emptyset) \wedge (A^I_{S_k} = \{b\})$ ***do***
13: $\quad R := R \triangleleft_a S_k$
14: ***end for***
15: $AlignedProtocols := AlignedProtocols \cup \{R\}$
16: *return* $AlignedProtocols$

It is worth to note that after having performed the procedure Alignment, P and R have the same alphabet of actions, except for possible third-party and hidden actions. Hereafter, given a set W of communication mediators and a protocol P, we denote with $P_{\triangleleft W}$ the protocol P whose alphabet has been aligned by means of the mediators in W.

Coming back to our case study, after having performed the alphabet alignment step, the protocol of BS becomes the one shown in Figure 4. The protocol of MC does not change.

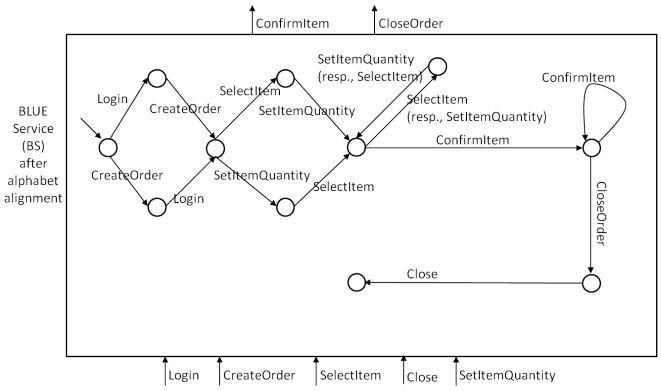


Fig. 4. BS after the alphabet alignment step

However, communication mediators are not able to solve all mismatches such as coordination mismatches. For instance, although the two protocols shown in Figures 1 and 4 share the same alphabet of actions, their interaction can still exhibit some mismatches. They are due to (i) messages sent/received in a different order (see the sequences made of $ConfirmItem$ and $CloseOrder$); (ii) third-party messages ($PayThirdParty$); and (iii) extra/missing sends corresponding to redundant messages (possibly also coming from looping/cyclic behavior, e.g., $SelectItem$ and $SetItemQuantity$). Thus, in general, the construction of other mediators that can delegate/receive[2], consume, produce,

[2]To/from a third-party.

and reorder messages is required. We call this kind of mediators as *coordination mediators*.

B. Automatic Synthesis of Coordination Mediators

Given two protocols to be mediated, P and R, whose alphabets have been aligned, our method produces their respective sets of traces. P and R are prefix-closed and hence their sets of traces are finite. Furthermore, possible loops/cycles are considered k times (where k is a parameter of our synthesis algorithm). This means that our method produces finite sets of finite traces. We recall that, by Definition 2, possible hidden actions are not represented within a considered trace.

Hereafter, we denote with $\overline{Tr(P)}^k$ the *set of traces* of P where, for each trace, cycles/loops are considered k times.

As it will be clear later in this section, once $\overline{Tr(P)}^k$ and $\overline{Tr(R)}^k$ have been generated, for all $(t_P, t_R) \in \overline{Tr(P)}^k \times \overline{Tr(R)}^k$, our method tries to synthesize a coordination mediator that makes the protocols corresponding to t_P and t_R able to interoperate. If no mediator has been synthesized, then a modular connector for P and R does not exist. Otherwise, a non-empty set of coordination mediators is produced. Indeed, considering all pairs in $\overline{Tr(P)}^k \times \overline{Tr(R)}^k$ is not needed. It is sufficient to consider only the subset of pairs of *semantically related* traces. Traces t_P and t_R are semantically related if every action that does not belong to their set of common actions is a third-party action.

Definition 11 (*Pairs of Semantically Related Traces*) *Given P and R to be mediated, let k be an integer such that $k > 0$, $(t_P, t_R) \in \overline{Tr(P)}^k \times \overline{Tr(R)}^k$ is a* pair of semantically related traces *if and only if the following conditions hold:*

- $a \in A_{t_P} \setminus A_{t_R} \Rightarrow \nexists t \in \overline{Tr(R)}^k \setminus t_R : \overline{a} \in A_t$;
- $a \in A_{t_R} \setminus A_{t_P} \Rightarrow \nexists t \in \overline{Tr(P)}^k \setminus t_P : \overline{a} \in A_t$.

We denote with $\Pi(P, R, k)$ the k-bounded set of pairs of semantically related traces for P and R.

Before continuing our formalization, Definition 11 deserves some discussion. It formalizes a *strong* notion of *semantic co-relation* for traces. For instance, let us consider traces t_P and t_R as shown in Figure 5. They are not semantically related because both actions e and f are in the set of common actions of P and R and, hence, cannot be considered as third-party actions, although are not common for t_P and t_R.

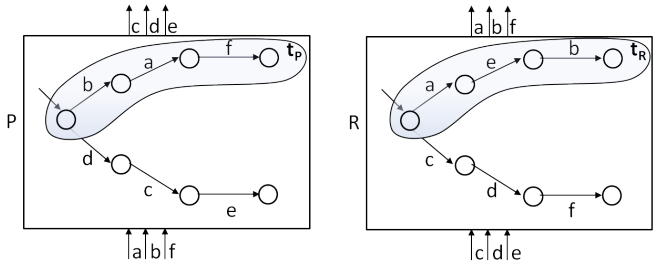


Fig. 5. t_P and t_R: two traces that are not semantically related

However, note that a way of mediating t_P and t_R exists, e.g., by re-ordering a and b and producing e and, afterwards, f. This mediation logic is correct only if e and f are independent from a and b, and a *causality dependency* is defined for f and e, e.g., the production of f depends on the production of e. Otherwise (e.g., the production of e depends on the production of f), it could result in an inconsistent mediation. Note that the relations defined in the protocol ontology do not necessarily represent all the possible causality dependencies. They are those related only to the semantics and structure of the exchanged data. That is, causality dependencies related to the control structure, and hence depending on the protocol semantics, could not be represented. Thus, despite the existence of a protocol ontology, without further information about protocol actions and their causality dependencies, one cannot in general know how to correctly mediate t_P and t_R. The safest choice that one can take, while synthesizing coordination mediators, is to discard this pair of traces. However, to avoid inconsistencies, discarding a pair of traces means that the mediator being synthesized has to consume all the outputs from the two traces. In other words, although this strong notion of semantic co-relation leads to synthesize correct modular connectors (i.e., that do not introduce inconsistencies), the synthesized connector might constrain too much the set of interactions on which the two protocols are made able to interoperate, hence possibly under-using them.

To avoid this problem we can consider a *weak* notion of semantic co-relation. This notion assumes the existence of a specification of the desired *coordination protocol* that the coordination mediator to be synthesized should satisfy. For instance, by referring to the example depicted in Figure 5, the coordination protocol specification could express that e *must always precedes* f or, equivalently, that f *must be always eventually performed after* e. For space reasons, we do not formalize this weak notion of semantic co-relation and, hereafter, we consider only the notion formalized by Definition 11.

Continuing the formalization of our synthesis method, for each pair $(t_P, t_R) \in \Pi(P, R, k)$, the method computes the so called *difference pair* (t'_P, t'_R). t'_P (resp., t'_R) is a sub-trace of t_P (resp., t_R) representing, in a single sequence, the sequences of actions in which t_P (resp., t_R) differs from t_R (resp., t_P). Due to the alphabet alignment, finding a coordination mediator for t_P and t_R means finding a coordination mediator for t'_P and t'_R. Since t_P and t_R are semantically related, their possible hidden actions have been removed, and their loops/cycles are considered k times, t'_P and t'_R can be different for three reasons only: (i) they have unshared actions corresponding to input/output third-party actions; (ii) they exhibit extra/missing sends corresponding to redundant messages, possibly also coming from looping/cyclic behavior; and (iii) they have complementary shared actions that appear in a different order. By means of the coordination mediators to be synthesized, the first ones should be received by a third-party (resp., an NS) and delegated to the receiving NS (resp., third-party), the second ones should be produced/consumed, and the third ones

should be reordered.

Definition 12 (*Coordination Mismatches Resolution*) *Let W be the set of synthesized communication mediators for protocols P and R, and valid protocol ontology O. Let k be the bound considered for the length of possible loops/cycles in $P_{\triangleleft W}$ and $R_{\triangleleft W}$. The algorithm for the* coordination mismatches resolution *step (i.e., the automatic synthesis of coordination mediators) is as follows:*

procedure *CoordinationMismatchesResolution*
input: $P_{\triangleleft W}$, $R_{\triangleleft W}$, k
output: $CoordMediators$

1: $CoordMediators := \emptyset$
2: ***for* each** $(t_{P_{\triangleleft W}}, t_{R_{\triangleleft W}}) \in \Pi(P_{\triangleleft W}, R_{\triangleleft W}, k)$ ***do***
3: *computes the difference pair* $(t'_{P_{\triangleleft W}}, t'_{R_{\triangleleft W}})$ *of* $(t_{P_{\triangleleft W}}, t_{R_{\triangleleft W}})$
4: ***for* each** $a \in (A_{t'_{P_{\triangleleft W}}} \setminus A_{t'_{R_{\triangleleft W}}}) \cup (A_{t'_{R_{\triangleleft W}}} \setminus A_{t'_{P_{\triangleleft W}}})$ ***do***
5: $CoordMediators := CoordMediators \cup \{ThirdParty(a)\}$
6: *remove a from either $t'_{P_{\triangleleft W}}$ or $t'_{R_{\triangleleft W}}$*
7: ***end for***
8: ***if*** $|t'_{P_{\triangleleft W}}| \neq |t'_{R_{\triangleleft W}}|$ ***then***
9: ***for* each** $a \in A_{t'_{P_{\triangleleft W}}} \cup A_{t'_{R_{\triangleleft W}}}$ *that appears more than once in either $t'_{P_{\triangleleft W}}$ or $t'_{R_{\triangleleft W}}$* ***do***
10: $CoordMediators := CoordMediators \cup \{ExtraOrMissing(a)\}$
11: *remove a from either $t'_{P_{\triangleleft W}}$ or $t'_{R_{\triangleleft W}}$*
12: ***end for***
13: ***end if***
14: ***if*** $t'_{P_{\triangleleft W}} \neq t'_{R_{\triangleleft W}}$ ***then***
15: $CoordMediators := CoordMediators \cup \{Reorder(a, t'_{P_{\triangleleft W}}, t'_{R_{\triangleleft W}})\}$
16: $CoordMediators := CoordMediators \cup \{Reorder(a, t'_{R_{\triangleleft W}}, t'_{P_{\triangleleft W}})\}$
17: ***end if***
18: ***end for***
19: ***for* each** $(t_{P_{\triangleleft W}}, t_{R_{\triangleleft W}}) \notin \Pi(P_{\triangleleft W}, R_{\triangleleft W}, k)$ ***do***
20: $CoordMediators := CoordMediators \cup Discard(t_{P_{\triangleleft W}}, t_{R_{\triangleleft W}}, P_{\triangleleft W}, R_{\triangleleft W})$
21: ***end for***
22: *return* $CoordMediators$

where the $ThirdParty$, $ExtraOrMissing$, $Reorder$, and $Discard$ (sub-)procedures are defined as follows:

procedure *ThirdParty*
input: a
output: $Mediator$

1: $Mediator := [\![Trans(a, a')^*]\!]$
2: *return* $Mediator$

procedure *ExtraOrMissing*
input: a
output: $Mediator$

1: ***if*** *a is an input action* ***then***
2: $Mediator := [\![Prod(a)^*]\!]$
3: ***else***
4: $Mediator := [\![Cons(a)^*]\!]$
5: ***end if***
6: *return* $Mediator$

procedure *Reorder*
input: a, $t'_{P_{\triangleleft W}}$, $t'_{R_{\triangleleft W}}$
output: $Mediator$

1: $aList := []$
2: $pTuple := ()$
3: ***for* each** $a \in A_{t'_{P_{\triangleleft W}}}$ ***do***
4: ***if*** *a is an output action* ***then***
5: *add a as last element of $aList$*
6: *add $t'_{R_{\triangleleft W}}(a)$ as last element of $pTuple$*
7: ***end if***
8: ***end for***
9: $Mediator := Order(aList, pTuple, aList')$, *where* $aList' = [a'_1, \ldots, a'_n]$ *and* $aList = [a_1, \ldots, a_n]$
10: $Mediator := [\![Mediator^*]\!]$
11: *return* $Mediator$

procedure *Discard*
input: $t_{P_{\triangleleft W}}$, $t_{R_{\triangleleft W}}$, $P_{\triangleleft W}$, $R_{\triangleleft W}$
output: $Mediators$

1: ***for* each** $a \in A_{t_{P_{\triangleleft W}}} \cup A_{t_{R_{\triangleleft W}}} \wedge a \in A^O_{P_{\triangleleft W}} \cup A^O_{R_{\triangleleft W}}$ ***do***
2: $Mediators := Mediators \cup \{[\![Cons(a)^*]\!]\}$
3: ***end for***
4: *return* $Mediators$

Coming back to our case study, the synthesized coordination mediators are:
M_6=$[\![Trans(PayThirdParty, PayThirdParty')^*]\!]$;
M_7=$[\![Order([ConfirmItem, CloseOrder], (2,1), [ConfirmItem', CloseOrder'])^*]\!]$;
M_8=$[\![Order([SelectItem, SetItemQuantity], (2,1), [SelectItem', SetItemQuantity'])^*]\!]$;
M_9=$[\![(Prod(SelectItem))^*]\!]$;
M_{10}=$[\![(Prod(SetItemQuantity))^*]\!]$;
M_{11}=$[\![(Cons(ConfirmItem))^*]\!]$.
By referring to Definition 4, the modular connector for our case study is given by the following composition of coordination mediators: $M = M_6||\ldots||M_{11}$; plus the set $W = \{M_1, \ldots, M_5\}$ of communication mediator used for the alphabet alignment. As formally shown in the next section, under alphabet alignment, M is a correct connector meaning that both $\overline{P_{\triangleleft W}} \succeq M||(R_{\triangleleft W})$ and $\overline{R_{\triangleleft W}} \succeq (P_{\triangleleft W})||M$ hold.

V. Correctness and Connector Evolution

By taking into account hidden actions and denying broadcast communication (as done in [14]), the IA theory described in [15] can be used to synthesize, via a *quotient* operator $/$, a monolithic connector M such that $G \succeq P||M||R$, i.e., $M = G/(P||R)$. The formal definition of G is out of the scope of this work. For the purposes of this section, it is sufficient to say that G is an IPS, representing the *connected* system goal, which explicitly models three crucial conditions for correct communication and coordination: (*c1*) $P||M||R$ is not permitted to generate any inconsistencies; (*c2*) $P||M||R$ is only permitted to deadlock when all P, M, and R deadlock; and (*c3*) $P||M||R$ must satisfy the constraints imposed by the given protocol ontology.

Stating correctness of our synthesis method means showing that a modular connector M synthesized for protocols P and R is such that *c1*, *c2*, and *c3* hold, under alphabet alignment. However, note that *c2* and *c3* trivially hold by construction. In fact, when composing in parallel protocols, the only possibility to have "sink" states concerns scenarios

in which none of the protocols is willing to perform any action (*c2*); and communication mediators ensure alphabet alignment (*c3*). Thus, in this section, by considering W as the set of synthesized communication mediators, we focus on proving that $P_{\triangleleft W}||M||R_{\triangleleft W}$ is free from inconsistencies, i.e., it is defined (*c1*). To do this, we can exploit Definition 5, hence checking both $\overline{P_{\triangleleft W}} \succeq M||R_{\triangleleft W}$ and $\overline{R_{\triangleleft W}} \succeq P||M_{\triangleleft W}$.

Theorem 1 (*Correctness under alphabet alignment*) *Let M be a modular connector synthesized for aligned protocols $P_{\triangleleft W}$ and $R_{\triangleleft W}$, then the following properties hold:* (1) $\overline{P_{\triangleleft W}} \succeq M||R_{\triangleleft W}$, *and* (2) $\overline{R_{\triangleleft W}} \succeq P||M_{\triangleleft W}$.

Proof*: To prove* (1)*, we must prove that* (i) $A^I_{\overline{P_{\triangleleft W}}} \subseteq_{sem} A^I_{M||R_{\triangleleft W}} \wedge A^O_{M||R_{\triangleleft W}} \subseteq_{sem} A^O_{\overline{P_{\triangleleft W}}}$, *and* (ii) *there is an alternating simulation from $M||R_{\triangleleft W}$ to $\overline{P_{\triangleleft W}}$, with $s^0_{\overline{P_{\triangleleft W}}} \succeq s^0_{M||R_{\triangleleft W}}$. By construction, $A^I_M = A^O_{P_{\triangleleft W}} \cup A^O_{R_{\triangleleft W}}$ and $A^O_M \subseteq A^I_P \cup A^I_R$. Furthermore, $common(M, R_{\triangleleft W})$ is given by the union set of $A^O_{R_{\triangleleft W}}$ and a subset of $A^I_{R_{\triangleleft W}}$. Thus, by definition of complement operator for IPSs, it follows that both $A^I_{\overline{P_{\triangleleft W}}} \subseteq_{sem} A^I_{M||R_{\triangleleft W}}$ and $A^O_{M||R_{\triangleleft W}} \subseteq_{sem} A^O_{\overline{P_{\triangleleft W}}}$ hold. To prove (ii), we have to show that ii.1, ii.2, and ii.3 of Definition 5 hold, where P and R have been replaced with $\overline{P_{\triangleleft W}}$ and $M||R_{\triangleleft W}$, respectively. Let us assume that ii.1 does not hold. This means that $P \triangleleft W$ would perform an output action x that is not consumed by M. This is a contradiction since by construction M always consumes output actions from both $P \triangleleft W$ and $R \triangleleft W$. Analogously, if ii.2 would not hold, then $M||R_{\triangleleft W}$ would produce an output action that does not match any input action of $P \triangleleft W$. Again, this is a contradiction because $M||R_{\triangleleft W}$ produces output actions only when there is the need to match an input from $R_{\triangleleft W}$ with an input from $P_{\triangleleft W}$. ii.3 directly follows from the previous considerations hence recursively propagating the alternating simulation relation from $M||R_{\triangleleft W}$ to $\overline{P_{\triangleleft W}}$. The proof of* (2) *is analogous and hence, for space reasons, we omit it.* ■

Concerning the ability, for modular connectors, to evolve in response of possible changes, the most interesting scenario is related to changes at the level of the protocol ontology. In fact, syntactic changes at the level of the NSs' interface directly correspond to a relabeling of mediator inputs/outputs, and related concepts in the ontology. We recall that the synthesis of coordination mediators deals with sets of traces. Thus, changes at the protocol level imply to re-iter the synthesis step on the affected traces only, hence accordingly changing the corresponding mediators. However, in the worst case, i.e., all the traces of a protocol share at least one action, the entire synthesis step must be repeated.

As an example of a possible change at the level of the protocol ontology, let us go back to our case study and apply the following modification to the ontology O shown in Figure 3: $S_O = S_O \cup \{(AddItemToOrder, SelectItem), (AddItemToOrder, SetItemQuantity)\}$ and $A_O = A_O \setminus \{(AddItemToOrder, SelectItem, SetItemQuantity)\}$. Although simple, this change highlights the effectiveness of our decomposition with respect to supporting connector evolution. In fact, to address the applied change, it is sufficient to reason compositionally at the level of the algebra-based description of the modular connector M and related set W of communication mediators, instead of reasoning in terms of its underlying IA-based monolithic representation. In particular, by just looking at the mediators' interface, one can easily recognize that the communication mediator affected by the proposed change is M_3, while M_8, M_9, and M_{10} are the affected coordination mediators. Due to the fact that the aggregation tuple $(AddItemToOrder, SelectItem, SetItemQuantity)$ has been removed by A_O, M_3 is removed as well. In place of it two communication mediators, $M_{3.1} = [\![(Split(SelectItem, [AddItemToOrder, z]) \odot Cons(z))^*]\!]$ and $M_{3.2} = [\![(Split(SetItemQuantity, [AddItemToOrder, k]) \odot Cons(k))^*]\!]$, are synthesized due to the addition to S_O of the above considered subsumption tuples. Furthermore, we recall that the IPS of BS has been modified in order to align its alphabet to the one of MC. To reflect the change on the performed alphabet alignment, a trace in $\overline{Tr(MC)}^k$ that contains $SelectItem$ and/or $SetItemQuantity$ is modified by considering the following substitution: $\{AddItemToOrder/SelectItem, AddItemToOrder/SetItemQuantity\}$. Analogously, a trace in $\overline{Tr(BS)}^k$ that contains either the sequence $\langle SelectItem\ SetItemQuantity\rangle$ or $\langle SetItemQuantity\ SelectItem\rangle$ is modified by replacing any of these sequences with $AddItemToOrder$. According to the new alphabet alignment, M_8 is removed and in place of both M_9 and M_{10} the coordination mediator $[\![(Prod(AddItemToOrder))^*]\!]$ is synthesized. Note that, in the monolithic connector, $SelectItem$ and $SetItemQuantity$ would always appear one after the other and modifying the connector according to the applied change would mean to solve again the entire quotient problem.

VI. Related Work

Interoperability and mediation have been investigated in several contexts, among which integration of heterogeneous data sources [13], architectural patterns [22], patterns of connectors [12], Web services [10], [11], and algebra to solve mismatches [23]. For space reasons, we discuss only the works, from the different contexts, closest to our method.

The interoperability/mediation of protocols have received attention since the early days of networking. Indeed many efforts have been done in several directions including for example formal approaches to protocol conversion, like in [24], [25].

The seminal work in [4] is strictly related to the notions of mediator presented in this paper. Compared to our connector synthesis, this work does not allow to deal with ordering mismatches and different granularity of the languages (solvable by the split and merge primitives).

Recently, with the emergence of web services and advocated universal interoperability, the research community has been studying solutions to the automatic mediation of business processes [26], [27]. However, most solutions are discussed

informally, making it difficult to assess their respective advantages and drawbacks.

In [12] the authors present an approach for formally specifying connector wrappers as protocol transformations, modularizing them, and reasoning about their properties, with the aim to resolve component mismatches. In [28] the authors present an algebra for five basic stateless connectors that are symmetry, synchronization, mutual exclusion, hiding and inaction. They also give the operational, observational and denotational semantics and a complete normal-form axiomatization. The presented connectors can be composed in series and in parallel. Although these formalizations supports connector modularization, automated synthesis is not treated at all hence keeping the focus only on connector design and specification.

In [8], the authors use a game theoretic approach for checking whether incompatible component interfaces can be made compatible by inserting a converter between them which satisfies specified requirements. This approach is able to automatically synthesize the converter. In contrast to our method, their method needs as input a deadlock-free specification of the requirements that should be satisfied by the adaptor, by delegating to the user the non-trivial task of specifying that.

In other work in the area of component adaptation [7], it is shown how to automatically generate a concrete adaptor from: (i) a specification of component interfaces, (ii) a partial specification of the components interaction behavior, (iii) a specification of the adaptation in terms of a set of correspondences between actions of different components and (iv) a partial specification of the adaptor. The key result is the setting of a formal foundation for the adaptation of heterogeneous components that may present mismatching interaction behavior. Assuming a specification of the adaptation in terms of a set of correspondences between methods (and their parameters) of two components requires to know many implementation details (about the adaptation) that we do not want to consider in order to synthesize a connector.

VII. Conclusions and Future Work

In this paper, we formalized a method for the automated synthesis of modular connectors. A modular connector is structured as a composition of independent mediators, each of them corresponding to the solution of a recurring protocol mismatch. We have proven that our connector decomposition is correct and, by means of a case study, we have shown how it promotes connector evolution. An overall advantage of our approach with respect to the work in the state of the art (Section VI) is that our connectors have a modular software architecture organized as a composition of fundamentals mediation primitives. This supports connector evolution and automated generation of the connector's implementation code. In particular, we have recently released a first implementation (http://code.google.com/p/otf-connector/) of both the algebra primitives and the plugging operator. This implementation is based on the use of *Enterprise Integration Patterns* (http://www.eaipatterns.com/) and is developed through the *Apache Camel* framework (http://camel.apache.org/). Because of the way a modular connector is structured, the automatic generation of its actual code written in terms of our algebra implementation is viable and can be achieved with little effort. We have started to show, through its application to the real world case study presented in this paper, that our method supports connector evolution. As future work, we intend to carry out a rigorous empirical investigation to confirm the results reported in this paper. Another future research direction concerns the ability to infer the needed ontological information, out of the interface description of the two protocols, rather than assuming it as given.

References

[1] M. Weiser, "The computer for the twenty-first century," *Scientific American*, 1991.

[2] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *SIGSOFT Softw. Eng. Notes*, vol. 17, no. 4, 1992.

[3] R. Allen and D. Garlan, "A formal basis for architectural connection," *ACM Trans. Softw. Eng. Methodol.*, vol. 6, no. 3, 1997.

[4] D. M. Yellin and R. E. Strom, "Protocol specifications and component adaptors," *ACM Trans. Program. Lang. Syst.*, vol. 19, no. 2, 1997.

[5] P. Inverardi, R. Spalazzese, and M. Tivoli, "Application-layer connector synthesis," in *SFM*, 2011.

[6] V. Issarny, A. Bennaceur, and Y.-D. Bromberg, "Middleware-layer connector synthesis: Beyond state of the art in middleware interoperability," in *SFM*, 2011.

[7] C. Canal, P. Poizat, and G. Salaün, "Model-based adaptation of behavioral mismatching components," *IEEE Trans. Software Eng.*, 2008.

[8] R. Passerone, L. de Alfaro, T. A. Henzinger, and A. L. Sangiovanni-Vincentelli, "Convertibility verification and converter synthesis: two faces of the same coin," in *ICCAD*, 2002.

[9] J. Cortadella, A. Kondratyev, L. Lavagno, C. Passerone, and Y. Watanabe, "Quasi-static scheduling of independent tasksfor reactive systems," in *ICATPN*, 2002.

[10] F. Jiang, Y. Fan, and X. Zhang, "Rule-based automatic generation of mediator patterns for service composition mismatches," in *GPC*, 2008.

[11] X. Li, Y. Fan, J. Wang, L. Wang, and F. Jiang, "A pattern-based approach to development of service mediators for protocol mediation," in *WICSA*, 2008.

[12] B. Spitznagel and D. Garlan, "A compositional formalization of connector wrappers," in *ICSE*, 2003.

[13] G. Wiederhold and M. Genesereth, "The conceptual basis for mediation services," *IEEE Expert: Intelligent Systems and Their Applications*, vol. 12, no. 5, 1997.

[14] L. de Alfaro and T. A. Henzinger, "Interface automata," in *ESEC/FSE*, 2001.

[15] T. Chen, C. Chilton, B. Jonsson, and M. Kwiatkowska, "A compositional specification theory for component behaviours," in *ESOP*, 2012.

[16] M. Autili, C. Chilton, P. Inverardi, M. Kwiatkowska, and M. Tivoli, "Towards a connector algebra," in *ISoLA*, 2010.

[17] R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi, "Alternating refinement relations," in *CONCUR'98*, 1998.

[18] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[19] "CONNECT consortium. CONNECT Deliverable D3.3: Dynamic connector synthesis: revised prototype implementation. FET IP CONNECT EU project, FP7 GA n.231167, http://connect-forever.eu/."

[20] U. Aßmann, S. Zschaler, and G. Wagner, *Ontologies, Meta-Models, and the Model-Driven Paradigm*. Springer, 2006.

[21] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds., *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, 2003.

[22] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Chichester, UK: Wiley, 1996.

[23] M. Dumas, M. Spork, and K. Wang, "Adapt or perish: Algebra and visual notation for service interface adaptation," in *Business Process Management*, 2006.

[24] K. L. Calvert and S. S. Lam, "Formal methods for protocol conversion," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 1, 1990.

[25] S. S. Lam, “Correction to ”protocol conversion”,” *IEEE Trans. Software Eng.*, vol. 14, no. 9, 1988.
[26] R. Vaculín and K. Sycara, “Towards automatic mediation of OWL-S process models,” *Web Services, IEEE International Conference on*, 2007.
[27] R. Vaculín, R. Neruda, and K. P. Sycara, “An agent for asymmetric process mediation in open environments.” in *SOCASE*, 2008.
[28] R. Bruni, I. Lanese, and U. Montanari, “A basic algebra of stateless connectors,” *Theor. Comput. Sci.*, vol. 366, no. 1, 2006.

B Quantitative extension: computation of constraints

In this appendix, we briefly give the derivation of the constraints imposed on guards, invariants and co-invariants by means of the $\perp$-backpropagation technique. This procedure is applied to the product automaton shown in Figure 2.8.

The $\perp$-backpropagation procedure in our theory is implemented symbolically using assertional reasoning techniques (e.g. weakest pre-condition calculation). The idea is that we first identify the set of initial auto-$\perp$ and semi-$\perp$ states in the automaton, which are given by constraints, say α. Then we backpropagate α through output transitions and delay transitions in order to calculate new auto-$\perp$ and semi-$\perp$ states.

- For action transitions (say $a!$), which are used to find new auto-$\perp$ states, weakest pre-condition calculation suffices; it returns a constraint α' s.t. any state satisfying α' can take the $a!$ transition and go to another state satisfying α.
- For delay transitions, which are used to find new semi-$\perp$ states, in addition to the weakest pre-condition requirement, the calculation of α' also needs to guarantee that no input is enabled in any state of α'. Thus, a state s satisfying α' can delay-transit into another state s' satisfying α, and all delay successors of s before reaching s' must have all inputs disabled. The disabled inputs are calculated using an implicit $\perp$-removal sub-procedure (implemented as weakest precondition calculation on input transitions) interwoven into the $\perp$-backpropagation.

In our case study, initially we only have semi-$\perp$ states. For instance, location $BVT3ZN$ (i.e. the product state in which the component automata are in locations B, V, T, 3, Z, N resp.) is waiting for the input *play_frame* which must arrive before x' reaching 2 (specified by its co-invariant). For $BVT3ZN$, the set of semi-$\perp$ states is characterised as the constraint $x' > y + 1 \& x' > x \& x' > z - 8 \& bf == 0 \& b == ff$, which say that clock x' can reach 2 faster than any of x, y and z reaching 2, 1 and 10 resp. Thus, the co-invariant of $BVT3ZN$ will be violated before the violation of its invariant, which leads to $\perp$. Similarly we have $x' > x \& x' > z - 8 \& buf, bf == 0$ as β for $BVS3YN$.

Based on $x' > y + 1 \& x' > x \& x' > z - 8 \& bf == 0 \& b == ff$ as α, we can do $\perp$-elimination on *input?* by calculating the weakest pre-condition, which is $x' > 1 \& x' > x \& x' > z - 8 \& bf == 0 \& b == ff \& buf > 0$ as α'. Obviously all states satisfying α' can delay-transit into the violation of the $BVS3YN$ co-invariant (i.e. before the invariant violation) and no *play_frame?* is enabled along the way. Thus α' captures a set of new semi-$\perp$ states. We can combine α' and β for $BVS3YN$ and get the new constraints $x' > 1 \& x' > x \& x' > z - 8 \& bf == 0 \& b == ff$.

In this way, we propagate the constraints through the graph of the product automaton, finding new auto-$\perp$/semi-$\perp$ states and weakening the constraints at locations accordingly until the fixed-point is reached. The detailed step-by-step illustration of the constraint-based $\perp$-backpropagation for our case study is given in Figures B.2, B.3 and B.4.

The four key locations we are interested in are $BVS3YN$, $BVT3ZN$, $BUS3YN$ and $BUT3ZN$. Our strategy first iterates over $BVS3YN$ and $BVT3ZN$ to reach the local fixed-point, i.e. constraints $(0) \& (6)$ and $(d) \& (g)$. Then, locations $BUS3YN$ and $BUT3ZN$ are added and the iteration (over all four states) continues until the global fixed-points are reached, i.e. constraints $FP1$, $FP2$, $FP3$, $FP4$ in Figure B.1. The fixed-point constraint backpropagation in Figures B.2, B.3 and B.4 is done manually for illustrative purposes, but it is easy to see that the procedure can be automated in a relatively straightforward manner.

Once the fixed-points have been reached, we can:

1. strengthen the co-invariants for locations $BVS3YN$, $BVT3ZN$, $BUS3YN$ and $BUT3ZN$ in Figure 2.8 by removing those states characterised by the corresponding fixed-points $FP1$, $FP2$, $FP3$ and $FP4$ in Figure B.1; and
2. perform $\perp$-elimination, and modify the guard on the *start_play?* transitions to be $G1$, $G2$, $G3$ and $G4$ in Figure B.1 (so that proper delay can be observed before the playback is started).

After this, the inputs and outputs, along with invariants and co-invariants should be exchanged on the resulting automaton, in order to obtain the timed quotient.

Fixpoints:

FP1::= OR{A,E,J,P,N,[a],[f],[l],[o]}

FP2::= OR{I,II,XII,VI,VIII,[1],[3],[6],[8]}

FP3::= OR{7,9,III,IV,V,VII,IX,X,[2],[4],[5],[7],[9]}

FP4::= OR{n,s,D,H,O,L,M,Q,[e],[i],[j],[k],[n]}

Guards on the incoming 'start_play?' transitions:

G1::= u==0& (bf+5*r)>=5 or u==1&r<=1& bf+5*r>=10 or u==1&r>=2&bf>=4

G2::= u==0&y>0& (bf+5*r)>=4 or u==0&y==0& bf+5*r>=5 or
u==1&y>0&r<=1& (bf+5*r)>=9 or u==1&y==0&r<=1& bf+5*r>=10 or u==1&r>=2&bf>=4

G3::= b==tt&y>0& z+2*(bf+5*r)>=18 or b==tt&y==0& z+2*(bf+5*r)>=20 or
b==ff&y>0&r<=1& z+2*(bf+5*r)>=18 or b==ff&y==0&r<=1& z+2*(bf+5*r)>=20 or b==ff&r>=2&z+2*bf>=8

G4::= b==tt& z+2*(bf+5*r)>=20 or b==ff&r<=1& z+2*(bf+5*r)>=20 or b==ff&r>=2&z+2*bf>=8

Figure B.1: The fixed-points and guards

Figure B.2: Local fixed-point calculation

Figure B.3: Global fixed-point calculation I

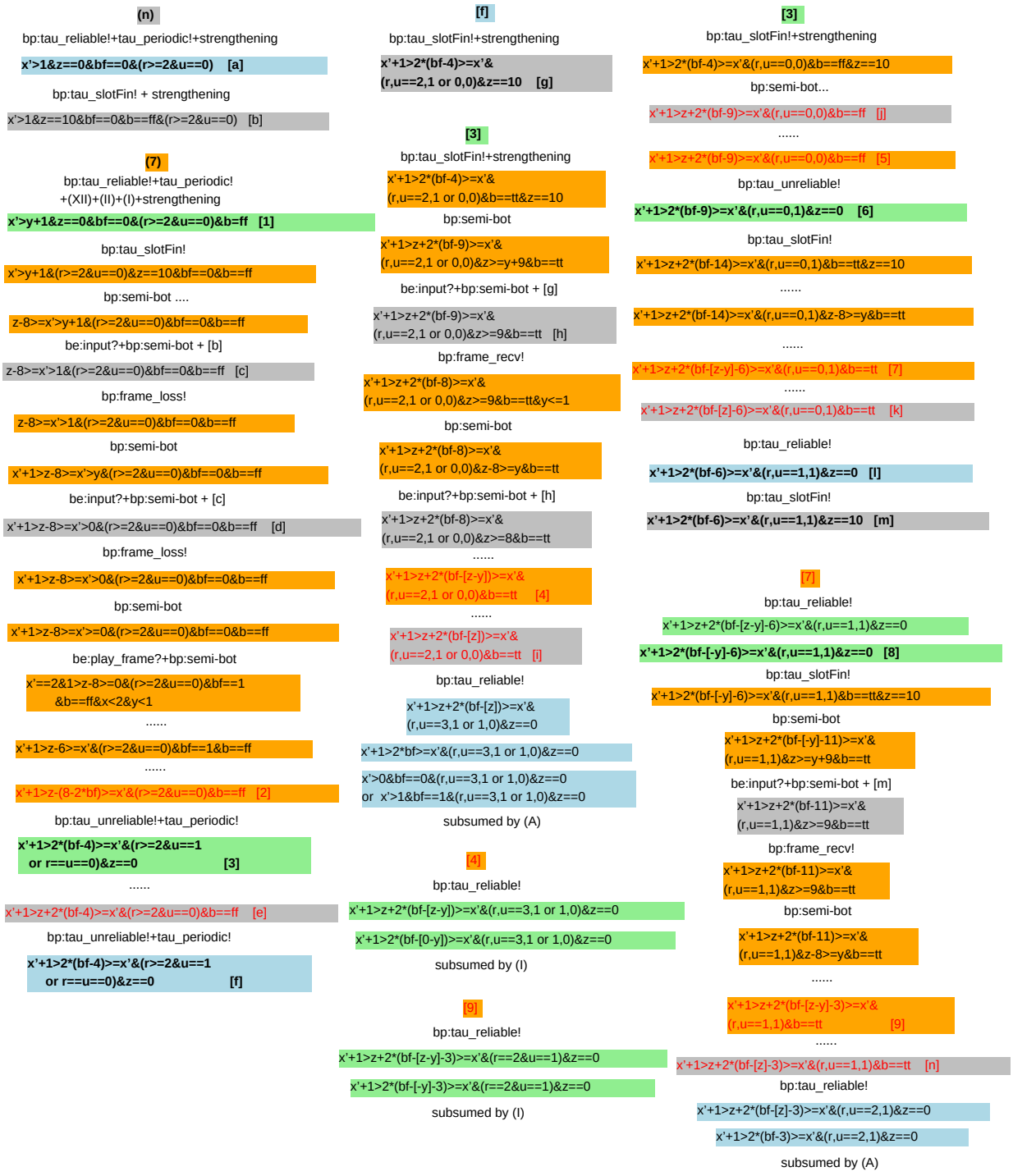


Figure B.4: Global fixed-point calculation II

Bibliography

[1] M. Abadi and L. Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, January 1993.

[2] M. Abadi and L. Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534, May 1995.

[3] R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126:183–235, April 1994.

[4] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, 2003.

[5] S. Bauer, A. David, R. Hennicker, K. Guldstrand Larsen, A. Legay, U. Nyman, and A. Wsowski. Moving from specifications to contracts in component-based design. In J. Lara and A. Zisman, editors, *Fundamental Approaches to Software Engineering*, volume 7212 of *Lecture Notes in Computer Science*, pages 43–58. Springer Berlin Heidelberg, 2012.

[6] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger, and K. G. Larsen. Contracts for System Design. Rapport de recherche RR-8147, INRIA, Nov. 2012.

[7] P. Bhaduri and S. Ramesh. Interface synthesis and protocol conversion. *Form. Asp. Comput.*, 20(2):205–224, March 2008.

[8] C. Canal, P. Poizat, and G. Salaün. Model-based adaptation of behavioral mismatching components. *IEEE Trans. Software Eng.*, 2008.

[9] T. Chen, C. Chilton, B. Jonsson, and M. Kwiatkowska. A Compositional Specification Theory for Component Behaviours. In H. Seidl, editor, *Programming Languages and Systems, Proc. 21st European Symposium on Programming (ESOP'12)*, volume 7211 of *Lecture Notes in Computer Science*, pages 148–168. Springer-Verlag, 2012.

[10] CONNECT consortium. CONNECT Deliverable D0.6: Annual progress and financial report. FET IP CONNECT EU project, FP7 grant agreement number 231167, http://connect-forever.eu/.

[11] CONNECT consortium. CONNECT Deliverable D2.2: Compositional algebra of connectors. FET IP CONNECT EU project, FP7 grant agreement number 231167, http://connect-forever.eu/.

[12] CONNECT consortium. CONNECT Deliverable D2.3: Rephrasing interoperability in terms of connector behaviours. FET IP CONNECT EU project, FP7 grant agreement number 231167, http://connect-forever.eu/.

[13] CONNECT consortium. CONNECT Deliverable D3.3: Dynamic connector synthesis: revised prototype implementation. FET IP CONNECT EU project, FP7 grant agreement number 231167, http://connect-forever.eu/.

[14] CONNECT consortium. CONNECT Deliverable D6.3: Experiment scenarios, prototypes and report Iteration 2. FET IP CONNECT EU project, FP7 grant agreement number 231167, http://connect-forever.eu/.

[15] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Timed I/O automata: a complete specification theory for real-time systems. In *Proc. 13th ACM International Conference on Hybrid systems: computation and control*, HSCC '10, pages 91–100. ACM, 2010.

[16] L. de Alfaro and T. Henzinger. Interface automata. In *Proceedings of the 8th European Software Engineering Conference and the 9th ACM Symposium on Foundations of Software Engineering*, pages 109–120, 2001.

[17] L. de Alfaro and T. A. Henzinger. Interface automata. *SIGSOFT Softw. Eng. Notes*, 26(5):109–120, September 2001.

[18] L. de Alfaro and T. A. Henzinger. Interface-based design. In M. Broy, J. Grünbauer, D. Harel, and T. Hoare, editors, *Engineering Theories of Software Intensive Systems*, volume 195 of *NATO Science Series II: Mathematics, Physics and Chemistry*, pages 83–104. Springer-Verlag, 2005.

[19] L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Timed interfaces. In A. Sangiovanni-Vincentelli and J. Sifakis, editors, *Embedded Software*, volume 2491 of *LNCS*, pages 108–122. Springer-Verlag, 2002.

[20] B. Delahaye, B. Caillaud, and A. Legay. Probabilistic contracts: a compositional reasoning methodology for the design of systems with stochastic and/or non-deterministic aspects. *Form. Methods Syst. Des.*, 38(1):1–32, Feb. 2011.

[21] J. Drissi and G. v. Bochmann. Submodule construction for systems of i/o automata. Technical report, M.B. JOSEPHS, H.K. KAPOOR / CONTROLLABLE DELAY-INSENSITIVE PROCESSES, 1999.

[22] M. Dumas, M. Spork, and K. Wang. Adapt or perish: Algebra and visual notation for service interface adaptation. In *Business Process Management*, 2006.

[23] B. Jonsson. Compositional specification and verification of distributed systems. *ACM Trans. on Programming Languages and Systems*, 16(2):259–303, 1994.

[24] K. G. Larsen, U. Nyman, and A. Wasowski. Interface input/output automata. In *FM 2006*, volume 4085 of *LNCS*, pages 82–97. Springer, 2006.

[25] X. Li, Y. Fan, J. Wang, L. Wang, and F. Jiang. A pattern-based approach to development of service mediators for protocol mediation. In *WICSA*, 2008.

[26] N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, September 1989.

[27] R. Passerone, L. de Alfaro, T. A. Henzinger, and A. L. Sangiovanni-Vincentelli. Convertibility verification and converter synthesis: two faces of the same coin. In *ICCAD*, 2002.

[28] J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, A. Legay, and R. Passerone. A modal interface theory for component-based design. *Fundam. Inform*, 108(1):119–149, 2011.

[29] B. Spitznagel and D. Garlan. A compositional formalization of connector wrappers. In *ICSE*, 2003.

[30] R. Vaculín, R. Neruda, and K. P. Sycara. An agent for asymmetric process mediation in open environments. In *SOCASE*, 2008.

[31] T. Verhoeff. *A Theory of Delay-Insensitive Systems*. PhD thesis, Dept. of Math. and C.S., Eindhoven Univ. of Technology, May 1994.

[32] G. Wiederhold and M. Genesereth. The conceptual basis for mediation services. *IEEE Expert: Intelligent Systems and Their Applications*, 12(5), 1997.