



Université
de Toulouse

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National Polytechnique de Toulouse (INP Toulouse)

Discipline ou spécialité :

Sureté de Logiciel et Calcul à Haute Performance

Présentée et soutenue par :

M. FLORENT CHEVROU

le mercredi 22 novembre 2017

Titre :

Formalisation of Asynchronous Interactions

Ecole doctorale :

Mathématiques, Informatique, Télécommunications de Toulouse (MITT)

Unité de recherche :

Institut de Recherche en Informatique de Toulouse (I.R.I.T.)

Directeur(s) de Thèse :

M. PHILIPPE QUEINNEC

MME AURELIE HURAUULT

Rapporteurs :

M. MOHAMED MOSBAH, INP BORDEAUX

M. STEPHAN MERZ, INRIA NANCY

Membre(s) du jury :

Mme CAROLE DELPORTE, UNIVERSITE PARIS 7, Président

Mme AURELIE HURAUULT, INP TOULOUSE, Membre

Mme BERNADETTE CHARRON-BOST, CNRS PARIS, Membre

Mme DELPHINE LONGUET, UNIVERSITE PARIS 11, Membre

M. PHILIPPE QUEINNEC, INP TOULOUSE, Membre

Abstract

Large computing systems are generally built by connecting several distributed subsystems. The way these entities communicate is crucial to the proper functioning of the overall composed system. An in-depth study of these interactions makes sense in the context of the formal development and verification of such systems. The interactions fall in two categories: synchronous and asynchronous communication. In synchronous communication, the transmission of a piece of information - the message - is instantaneous. Asynchronous communication, on the other hand, splits the transmission in a send operation and a receive operation. This makes the interleaving of other events possible and leads to new behaviours that may or may not be desirable. The asynchronous world is often viewed as a monolithic counterpart of the synchronous world. It actually comes in multiple models that provide a wide range of properties that can be studied and compared. This thesis focuses on communication models that order the delivery of messages: for instance, the “FIFO” models ensure that some messages are received in the order of their emission.

We consider classic communication models from the literature as well as a few variations. We highlight the differences that are sometimes overlooked. First, we propose an abstract, logical, and homogeneous formalisation of the communication models and we establish a hierarchy that extends existing results.

Second, we provide an operational approach with a tool that verifies the compatibility of compositions of peers. We mechanise this tool with the TLA⁺ specification language and its model checker TLC. The tool is designed in a modular fashion: the communicating peers, the temporal compatibility properties, and the communication models are specified independently. We rely on a set of uniform operational specifications of the communication models that are based on the concept of message history. We identify and prove the conditions under which they conform to the logical definitions and thus show the tool is trustworthy.

Third, we consider concrete specifications of the communication models that are often found in the literature. Thus, the models are classified in terms of ordering properties and according to the level of abstraction of the different specifications. The concept of refinement covers these two aspects. Thus, we model asynchronous point-to-point communication along several levels of refinement and then, with the Event-B method, we establish and prove all the refinements between the communication models and the alternative specifications of each given model. This work results in a detailed map one can use to develop a new model or find the one that best fits given needs.

Eventually we explore ways to extend our work to multicast communication that consists in sending messages to several recipients at once. In particular, we highlight the differences in the hierarchy of the models and how we modify our verification tool to handle this communication paradigm.

Résumé

Les systèmes informatiques sont construits par composition de plusieurs sous-systèmes répartis. La manière dont communiquent ces entités, ou pairs, joue un rôle clé dans la bonne marche du système composé. L'étude détaillée de ces interactions est donc essentielle dans le cadre de la vérification et du développement formel de tels systèmes. Ces interactions se décomposent en deux catégories: la communication synchrone et la communication asynchrone. La communication synchrone admet une transmission instantanée de l'information, le message, entre deux entités. La communication asynchrone, en revanche, prend en compte le découplage de la transmission du message en une opération d'envoi puis de réception avec la possibilité que des événements s'intercalent entre les deux donnant ainsi lieu à des variations de comportement, désirables ou non, des systèmes. Souvent considérée comme une entité monolithique duale du monde synchrone, le monde asynchrone se décline en réalité en de multiples modèles qui peuvent induire sur la communication une grande variété de propriétés qu'il convient de caractériser et comparer. Cette thèse se focalise sur les modèles de communication qui orchestrent l'ordre de délivrance des messages : par exemple les modèles dits FIFO qui assurent que certains messages sont reçus dans l'ordre dans lequel ils ont été émis.

Nous considérons des modèles de communication classiques de la littérature ainsi que des variations de ces modèles dont nous explicitons les différences parfois négligées. Dans un premier temps nous proposons une formalisation logique abstraite et homogène des modèles de communication considérés et nous les hiérarchisons en étendant des résultats existants.

Nous proposons dans un second temps une approche opérationnelle sous forme d'un outil de vérification de compositions de pairs que nous mécanisons à l'aide du langage de spécification TLA⁺ et du vérificateur de modèles TLC. Cet outil permet de spécifier des pairs communicants et des propriétés temporelles à vérifier pour les différents modèles de communication de façon modulaire. Pour cela, nous apportons un ensemble de spécifications uniformes et opérationnelles des modèles de communication basé sur la notion d'histoires de messages. Nous identifions et prouvons les conditions de leur conformité aux définitions logiques et validons ainsi la pertinence de notre outil.

Dans un troisième temps nous considérons des spécifications concrètes de nos modèles de communication, semblables à nombre de celles présentes dans la littérature. Nous disposons donc d'une hiérarchisation des modèles selon les propriétés d'ordre qu'ils garantissent mais également d'une autre hiérarchisation pour un modèle donné entre sa définition logique abstraite et ses implantations concrètes. Ces deux dimensions correspondent à deux dimensions du raffinement. Nous introduisons graduellement par raffinement la notion de communication asynchrone point à point et prouvons, grâce à la méthode Event-B, tous les liens de raffinement entre les différents modèles de communication et leurs déclinaisons. Nous offrons ainsi une cartographie détaillée des modèles pouvant être utilisée pour en développer de nouveaux ou identifier les modèles les plus adaptés à des besoins donnés.

Enfin, nous proposons des pistes d'extension de nos travaux à la communication par diffusion où un message peut être envoyé simultanément à plusieurs destinataires. En particulier, nous montrons les différences induites dans la hiérarchie des modèles et les adaptations à effectuer sur notre outil de vérification pour prendre en compte ce mode de communication.

Remerciements

Je souhaite particulièrement remercier Stephan Merz et Mohamed Mosbah, rapporteurs de cette thèse, qui ont consacré leur temps à la lecture détaillée du présent manuscrit et m'ont transmis leurs remarques. Je remercie également Bernadette Charron-Bost, Carole Delporte, et Delphine Longuet d'avoir accepté de faire partie du jury et de venir assister à ma soutenance.

Je tiens bien entendu à remercier mes directeurs de thèse : Aurélie Hurault et Philippe Quéinnec, qui m'ont offert en tout premier lieu l'opportunité de travailler avec eux sur ce sujet. Au cours de ces quelques années partagées, ils ont fait preuve d'une disponibilité sans faille et pris à cœur leurs rôles d'encadrants. Je les remercie pour leur implication, leur soutien, leur patience souvent, leur indulgence parfois, et la qualité de nos échanges.

Il serait difficile de dresser une liste exhaustive des personnes avec qui ce fut un plaisir d'évoluer au quotidien. Je salue donc tous les membres de l'équipe ACADIE pour leur aide et leurs conseils, en particulier Xavier Thirioux et Philippe Mauran, les nombreux collègues avec qui j'ai eu l'occasion de travailler en enseignement, ainsi que l'équipe des secrétaires qui font rimer administration avec efficacité et bonne humeur.

Impossible enfin de terminer sans citer ceux avec qui j'ai partagé un bureau et le plus clair du quotidien de ces dernières années. Un grand merci à Guillaume et Mathieu pour l'excellente ambiance, nos longues discussions, et nos légendaires dérapages de fin de semaine ; une pensée pour Kahina et nos premières années de thèse ainsi que pour Adam qui assure déjà brillamment la relève.

Acknowledgements

I wish to thank Professor Shin Nakajima for his warm welcome and precious advice during my internship at the National Institute of Informatics in Tokyo.

Contents

1	Introduction	13
1.1	Problem	13
1.1.1	Synchronous Communication	13
1.1.2	Asynchronous Communication	14
1.2	Contributions	15
1.3	Outline of this Work	15
1.3.1	Distributed Systems and Formal Verification	15
1.3.2	Point-to-point Communication	16
1.3.3	Group Communication	17
I	Distributed Systems and Formal Verification	19
2	Distributed Systems	21
2.1	Message-Passing Communication	21
2.2	Point-to-Point Communication	25
2.3	Group Communication	25
2.4	Asynchronous Communication Models	27
2.4.1	<i>Fully Asynchronous</i> Communication	27
2.4.2	<i>FIFO 1-1</i> Communication	27
2.4.3	<i>Causal</i> : Causally Ordered Communication	28
2.4.4	<i>FIFO n-1</i> Communication	31
2.4.5	<i>FIFO 1-n</i> Communication	31
2.4.6	<i>FIFO n-n</i> Communication	35
2.4.7	<i>RSC</i> : Realisable with Synchronous Communication	37
2.4.8	Summary of the Asynchronous Communication Models	37
2.5	Hierarchy	38
3	State of the Art	45
3.1	Description of Distributed Systems	45
3.1.1	Transition Systems	45
3.1.2	I/O Automata	46
3.1.3	Message Sequence Charts	46
3.1.4	Choreographies and Compatibility Checking	47
3.1.5	Process Calculi	48
3.2	Asynchronous Communication in Distributed Systems	51
3.2.1	Hierarchy of Ordering Paradigms	51
3.2.2	Hierarchy of Operational Communication Models	52

3.2.3	Realisability with Synchronous Communication	52
3.2.4	Summary	52
3.3	Formal Verification	53
3.3.1	Proof Assistance	54
3.3.2	Correct-by-Construction Design of Distributed Systems	54
II	Point-to-Point Communication	57
4	Compatibility Checking of Communicating Peers	59
4.1	Description and Formalisation of the Framework	61
4.1.1	Channels	61
4.1.2	Specification of Compositions of Peers	61
4.1.3	Specification of Communication Models	63
4.1.4	Overall Product System	64
4.1.5	Compatibility Checking	66
4.1.6	Specification of Communication Models with Message Histories	69
4.1.7	Specification of Capped Asynchronous Communication	73
4.2	Conformance to the Specifications	74
4.2.1	Correctness	76
4.2.2	Completeness	80
4.3	Conclusion	87
5	Mechanised Compatibility Checking with TLA⁺	89
5.1	The TLA ⁺ Specification Language	89
5.2	Organisation and Structure of the TLA ⁺ Modules	90
5.3	User-Friendly Automations	92
5.3.1	Alternate Specification of a Peer using a CCS Term	92
5.3.2	Faulty Reception Completion	96
5.3.3	Composite Communication Models	97
5.4	Examples and Results	100
5.4.1	Detailed Example: The Examination Management System	100
5.4.2	Practical Example: The Client-Controller-Application System	102
5.4.3	Advanced Usage of Composite Models: the Video Stream	102
5.5	Optimised Communication Models	104
5.5.1	Reduction to Finite State Spaces by Purging Histories	105
5.5.2	Dedicated Optimised Implementations	105
5.6	Benchmarking	105
5.6.1	Scenario	105
5.6.2	Analysis	107
5.7	Conclusion	108
6	A Menagerie of Refinements	109
6.1	Introduction	109
6.2	Distributed Systems	110
6.2.1	Distributed Executions	110
6.2.2	Event-B	110
6.2.3	From Events to Distributed Executions	111
6.2.4	Summary	116
6.3	Abstract Communication Models	116

6.3.1	Specifications of the Communication Models	116
6.3.2	Reduction of Non-Determinism	118
6.3.3	Proofs and Invariants	118
6.4	History-based Communication Models	120
6.4.1	Specifications with Histories	120
6.4.2	Refinement of Events by Histories	121
6.4.3	Preservation of the Hierarchy	126
6.5	Concrete Communication Models	128
6.5.1	Refinement with Counters of Messages	128
6.5.2	Refinement with Queues of Messages	130
6.5.3	Logical Clocks	130
6.6	Additional Remarks	132
6.6.1	Proof Effort	132
6.6.2	Deadlock Freedom	133
6.6.3	Previous Work in TLA ⁺	133
6.6.4	Utility of the Hierarchies	133
6.6.5	Localisation	134
6.7	Conclusion	135
III	Group Communication	137
7	Multicast Communication	139
7.1	Extension of the Hierarchy of Communication Models	139
7.1.1	Totally Ordered Multicast Distributed Executions	139
7.1.2	One-to-All Communication	141
7.2	Towards a Mechanised Framework	146
7.2.1	Lifespan of a Message in the Network of Messages In-Transit	146
7.2.2	Preventing a Peer from Receiving the Same Message Twice	147
7.2.3	Specification of the Interest	147
7.2.4	Point-to-Point and One-to-All Communication	148
7.2.5	Organisation and Structure of the TLA ⁺ Modules	148
7.2.6	Communication Models	149
7.2.7	A Composite Communication Model	155
7.2.8	Limitations	158
7.2.9	Addressing the Issue	159
7.3	Conclusion	160
IV	Conclusion	161
8	Conclusion and Future Work	163
8.1	Results and Practical Benefits	163
8.2	Future Work	164
8.2.1	Wider Range of Communication Models	164
8.2.2	Equivalence between Communication Models with regard to Compatibility	165

Chapter 1

Introduction

Computing systems usually consist of a collection of individual components that interact with each other to fulfill a common purpose: for instance computing a result or making a consensual decision. The components do not have direct access to a shared knowledge and must exchange information between each other to achieve their goal. Interestingly, everyday situations are often reminiscent of the difficulty of this task. Friends have indeed always struggled to agree on a meeting time for a night out to the movies; some never receive the last update in time; others are confronted to conflicting information; and overly polite friends have sometimes missed the beginning of the show waiting for each other to cross the doorstep of the theater first. All these hassles find echoes in actual *distributed systems* as challenging incompatibilities. Guaranteeing a collection of components is compatible is indeed far from trivial. With the advent of cloud computing, the Internet of Things, and the resulting interlinking of computerised objects spanning from large data centers to the pettiest kitchen appliances, the *distributed systems* are gaining even stronger influence and sway on our lives. As a consequence, ensuring their proper functioning is crucial. Failing to do so can result in physical, environmental, economical, or even life-threatening complications. Many critical systems are indeed inherently distributed: classic examples include sensors and calculating units in aircrafts or autonomous cars, emergency telecommunication networks, or parallel computing of medical data.

Formal methods propose to meet such demanding safety and performance requirements by instrumenting mathematical structures and proofs to describe, model, and reason on the design of the systems in order to minimise the risks of bugs and the burden of troubleshooting. They offer to certify systems actually conform to a collection of specifications and properties of well-behaviour. In that regard, this work contributes to the formalisation of a specific domain of interactions in distributed systems called asynchronous interactions.

1.1 Problem

In distributed systems, a collection of entities called peers exchange information by sending messages to each other. The semantics of the transmission of messages is however not unique.

1.1.1 Synchronous Communication

A simple viewpoint considers this transmission to be instantaneous. This is called *synchronous communication*. Extensive work has been carried out to study this model of interaction in distributed systems. The clarity and minimalism of synchronous communication is an asset in

formal reasoning that eases modelling and proof. In some cases, synchronous communication may be an appropriate rough description that leads to satisfactory results. Nevertheless, it fails to capture simple situations. As an example, consider Alistair who goes to the post office to send his friend Roberta a parcel containing a present for her birthday. The next day, he realises he forgot to include a birthday card and goes back to the post office to mail her one. Roberta receives the card first because of how the transporter handles large parcels. This scenario cannot be described by synchronous communication (without explicitly modelling the post office). Synchronicity implies Roberta receives the parcel at the time Alistair sends it. Hence, there are only two possible outcomes: the parcel is transmitted first, or the card is transmitted first, none of which accurately models the considered scenario that may yet occur in practice.

1.1.2 Asynchronous Communication

Asynchronous communication splits the transmission of a message in two separate events: the emission and the delivery that do not happen simultaneously. Taking the asynchrony into account allows to describe the previous situation. This opens the way for any possible interleaving of the communication events. Yet, asynchronous communication is not to be mistaken for a single interaction model. It actually covers a wide range of interaction models. Consider a new policy at the post service that now guarantees the letters and parcels sent from Alistair's post office are always delivered in the order they have been dropped in the letterbox. This new policy still describes asynchronous communication: Alistair and Roberta may phone while the parcel and the letter are still in transit because the transmission is not instantaneous. Yet, it is not the same asynchronous interaction model that allowed the previous scenario to happen, it is in fact characterised by a property on the order of the deliveries. If the presented scenario were undesirable, the choice of the communication model would be crucial when it comes to the design of a system that prevents it. Hence, knowing, describing, and comparing these models of asynchronous interactions is key to the quest for well-behaved distributed systems.

There are currently several issues that arise in the study of asynchronous interactions:

Clarity The asynchronous communication models are unfortunately often a source of confusion: various asynchronous communication models wrongly fall under the generic term “asynchronous” although they actually incorporate particular properties of the communication in addition to the decoupling of the emissions and deliveries. “FIFO” is another term used in the literature to describe a whole family of communication models that exhibit fundamental differences.

Exhaustiveness and Consistency There is no standard for the specification of common asynchronous communication models. Structural descriptions and operational descriptions co-exist side by side. They branch again into specifications based on different data structures. There is a lack of exhaustiveness and consistency which makes it difficult to compare the models.

Comparison The different options in the choice of a communication model and its formal specification are seldom motivated, let alone the comparison of these models and specifications. The existing studies of the relations between the communication models lack popular communication models, in particular in the “FIFO” family of communication models. A clear overview of the relations between the communication models helps

Considering the role of the communication model in the compatibility of communicating peers and the consequences of unsafe systems, these caveats pose challenges it is relevant to tackle.

1.2 Contributions

This work addresses the three challenges in the formalisation of asynchronous interactions bearing in mind the ultimate purpose of this undertaking: supporting the design of trustworthy distributed systems. The main contributions are:

- uniform, generic, and simple structural definitions of the asynchronous communication models;
- a generic approach for both point-to-point communication and group communication where messages can be sent to several peers at a time;
- uniform, generic, and simple operational specifications of the asynchronous communication models;
- an extensive and mechanically certified cartography of the relations between the asynchronous communication models and the alternate specifications of each communication model;
- a full-featured certified framework for compatibility checking of compositions of peers in point-to-point communication;
- a practical and user-friendly mechanisation of this framework;
- an extension of the contributions to group communication.

1.3 Outline of this Work

This work is articulated in three parts. The first one provides a detailed overview of the domains and concepts on which the thesis relies: namely distributed systems, asynchronous communication, and formal methods. The second and third parts explore two aspects of communication: point-to-point and multicast communication. In the second part, we design, formalise, mechanise, and review extensive frameworks dedicated to the in-depth study of point-to-point communication and compatibility in point-to-point distributed systems. In the third part, we propose to extend previous contributions to multicast communication. Eventually, after a thorough review of these propositions, we conclude on the contributions and glance forward to perspectives and future work.

1.3.1 Distributed Systems and Formal Verification

In Chapter 2, we introduce the concepts of message-passing communication in distributed systems with a focus on asynchronous communication and the different declinations, the communication models, that we are to study throughout the entire thesis. The three main contributions of this chapter are the following:

- We consider additional communication models that are rarely taken into account in existing work. Sometimes, these models are not distinguished from each other and fall under the generic term “FIFO” although they carry fundamental differences. We show these differences and provide a formal base that allows to easily highlight and compare them.

- We unify the formalisation of the seven communication models we consider using the notion of distributed executions. The distributed executions are partially ordered sets of internal or communication events that permit to describe distributed systems as a whole. It is well adapted to the definition of ordering policies on message deliveries in the communication models.
- Thanks to the uniform definitions of the communication models, we prove how they relate to each other depending on the strength of the underlying ordering policy and we deduce an overall hierarchy for both point-to-point communication *and* multicast communication. Existing results on the matter seldom take multicast communication into account nor the different declinations of FIFO communication.

The formal ground of Chapter 2 and the definition of the different message-ordering paradigms serve as a reference for the contributions in the following chapters.

Chapter 3 explores the context of the thesis and previous work carried on the formalisation and study of asynchronous communication in distributed systems. First, we browse different formalisms designed or adapted to the specification of distributed systems among which transition systems and automata, message sequence charts, or process calculi. For each one of them, we identify which characteristics are of interest to our work. We then focus on existing results on the study and comparison of the ordering paradigms. Finally, we explore the different aspects of formal verification techniques that play central roles in this work.

1.3.2 Point-to-point Communication

Chapter 4 is about the formal verification of the compatibility of compositions of peers in asynchronous point-to-point communication. The study of the compatibility of communicating peers aims at illustrating the key influence of the communication model on the well-behaviour of distributed systems and the importance of a strict formal framework dedicated to compatibility checking. This chapter motivates the different choices made in the design and formalisation of such a framework. The framework we propose allows to specify individual peers as well as communication models with transition systems that interact and yield to an overall system over which temporal compatibility properties are to be checked. The main contributions of this chapter are the following:

- We formalise a full stack framework for the verification of communicating peers that remains as generic and simple as possible. It is based on transition systems and a custom product operation that makes the peers interact with the communication medium.
- We consider channel-based communication where channels are not limited to one sender and one receiver.
- We formalise uniform operational specifications of the seven communication models we study. They are a compromise that describes models that can reasonably be considered for implementation yet abstract enough to ease their formal study and comparison.
- We prove these specifications conform to the reference logical definitions of the ordering policies from Chapter 2 thus ascertaining a high level of trust in the framework.

In the following chapter, we mechanise the entire framework with the TLA⁺ specification language and the TLC model checker. The result is a fully automated and modular tool that retains every aspect of the theoretical description, including the formal proofs of conformance to

the logical descriptions, among with simplifications of the peer specification process. It provides compatibility results and counterexamples for a given set of peers, a communication model, and a compatibility criterion. Practical examples illustrate these features and we eventually provide a performance review that compares alternate optimised specifications of the communication models.

In Chapter 6, we extend results about how the ordering policies relate to each other in order not only to take into account the logical descriptions of the communication models but the different descriptions that fit a given model as well. We rely on refinement and a mechanisation of the proofs with the Event-B method. This chapter aims at drawing the big picture that helps identifying the most adapted communication model to a given situation and the most adapted specification of that communication model among the variety of descriptions that correspond to different levels of abstraction. The contributions of this chapter are:

- An extensive library of operational specifications of the different models including new concrete practical specifications specified using the Event-B method.
- The use of concretisation refinement to prove that concrete specifications of the models still conform to the reference definitions and the levels of abstractions in-between.
- The use of refinements for simulation to prove the hierarchy of the communication models holds at more concrete levels of specification.
- The mechanisation of these proofs using the Event-B method.

1.3.3 Group Communication

In Chapter 7, we extend point-to-point contributions to multicast communication. We add the study of total ordering, a property of multicast communication, to the existing hierarchy of communication models. We also propose a new mechanised framework in TLA^+ that handles group communication and point-to-point communication in a generic approach.

Part I

Distributed Systems and Formal Verification

Chapter 2

Distributed Systems

A distributed system is composed of peers that interact with each other by exchanging messages. In asynchronous communication, the messages are sent and received during two distinct communication events and transmitted according to rules dictated by a communication model. This chapter introduces all these concepts and serves as a reference throughout the remainder of this work. It lays down the formal base for the description of asynchronous interactions, the distributed executions, and then presents seven common communication models. The seven communication models are eventually compared to establish a global hierarchy.

2.1 Message-Passing Communication

Let \mathcal{P} be an enumerable set of communicating peers and \mathcal{M} an enumerable set of messages. In message-passing communication, the transmission of a piece of information, the message, between two peers, corresponds to a couple of communication events: namely send and receive. An event occurs on a peer. A communication event can be a send event or a receive event and it is associated to a message. Let $\mathcal{L} \triangleq \{\text{Send}, \text{Receive}, \text{Internal}\}$ the set of event labels. Events do not carry information about the type of event (send, receive, or internal), the message, and the peer where it occurs. A distributed execution is a partially ordered set of events with labelling functions that provide the information and give meaning to those events. It describes which messages are sent and received by the peers in the system. The partial order is usually named the causal order [Lam78, Ray13]. It abstracts independent events.

Definition 1 (Distributed Execution). *A distributed execution $(E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \text{com}, \text{peer}, \text{mes})$ is a partially ordered set with labelling functions where E is an enumerable set of events, and $\text{com}, \text{peer}, \text{mes}$ are labelling functions.*

- $\text{com} \in E \rightarrow \mathcal{L}$ provides the nature of an event: an internal event `Internal` or a communication event `Send` or `Receive`.
- $\text{peer} \in E \rightarrow \mathcal{P}$ localises an event on a peer.
- $\text{mes} \in \text{com}^{-1}(\{\text{Send}, \text{Receive}\}) \rightarrow \mathcal{M}$ labels a communication event with a message.

Events occurring on the same peer are totally ordered.

$$\forall p \in \mathcal{P} : \leq_p \text{ is a total ordering on } \text{peer}^{-1}(\{p\})$$

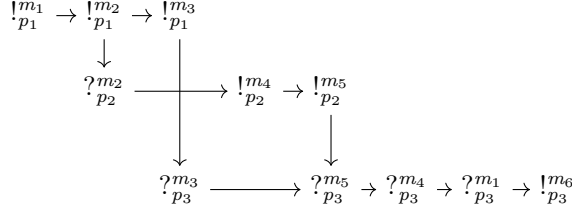


Figure 2.1: Example of a Distributed Execution $(E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \text{com}, \text{peer}, \text{mes})$. A send event of a message m by a peer p is denoted $!^m_p$ (an event $e \in E$ such that $\text{com}(e) = \text{Send} \wedge \text{mes}(e) = m \wedge \text{peer}(e) = p$). A receive event of a message m on a peer p is denoted $?^m_p$ (an event $e \in E$ such that $\text{com}(e) = \text{Receive} \wedge \text{mes}(e) = m \wedge \text{peer}(e) = p$). A path from e_1 to e_2 means $e_1 \prec_c e_2$.

\prec_c is the causal partial order on E . It extends the total orderings on peers with the couples of send and receive events associated to the transmission of a message. It is the smallest (for the inclusion) order on E such that:

$$\forall e_1, e_2 \in E : e_1 \prec_c e_2 \Leftrightarrow \left(\begin{array}{l} \exists p \in \mathcal{P} : e_1 \leq_p e_2 \quad (\text{peer ordering}) \\ \vee \left(\begin{array}{l} \text{com}(e_1) = \text{Send} \\ \wedge \text{com}(e_2) = \text{Receive} \\ \wedge \text{mes}(e_1) = \text{mes}(e_2) \end{array} \right) \quad (\text{transmission of a message}) \\ \vee \exists e \in E : e_1 \prec_c e \wedge e \prec_c e_2 \quad (\text{transitivity}) \end{array} \right)$$

A receive event is preceded by a corresponding send event.

$$\forall e \in E : \text{com}(e) = \text{Receive} \Rightarrow \exists e' \in E : \text{com}(e') = \text{Send} \wedge \text{mes}(e') = \text{mes}(e) \wedge e' \prec_c e$$

Figure 2.1 illustrates a partially ordered set of events. It is a distributed execution. First, every receive event is preceded by a corresponding send event. For instance, the reception of m_1 on p_3 is preceded by the send of m_1 on p_1 . A send event does not need to be followed by an associated receive event: for example there is no receive event of message m_6 although there is a send event of m_6 on p_3 . Then, events occurring on the same peer are totally ordered. Each line in the diagram corresponds to a peer. The horizontal arrows account for the local total ordering of the events on each peer.

Definition 2 (Run). A run $(E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \prec_\sigma, \text{com}, \text{peer}, \text{mes})$ extends a distributed execution $(E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \text{com}, \text{peer}, \text{mes})$. (E, \prec_σ) is a totally ordered set of events, where \prec_σ is a linear extension of \prec_c :

$$\begin{aligned} \forall e, e' \in E : e \prec_c e' &\Rightarrow e \prec_\sigma e' \\ \forall e, e' \in E : e \neq e' &\Rightarrow e \prec_\sigma e' \vee e' \prec_\sigma e \end{aligned}$$

The set of runs that extend the distributed executions in a set Σ is denoted $\text{Runs}(\Sigma)$.

Assuming interleaving of independent events and no true concurrency, a run is a linear extension of a distributed execution.

Theorem 3 (Order Inclusion). Given a run $(E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \prec_\sigma, \text{com}, \text{peer}, \text{mes})$:

$$\forall e_1, e_2 \in E : \left(\begin{array}{l} (\exists p \in \mathcal{P} : e_1 \leq_p e_2) \Rightarrow e_1 \prec_c e_2 \\ \wedge e_1 \prec_c e_2 \Rightarrow e_1 \prec_\sigma e_2 \end{array} \right)$$

Proof. By Definition 1 of a distributed execution and Definition 2 of a run. \square

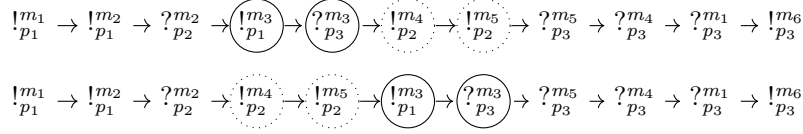


Figure 2.2: Two Runs $(E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \prec_\sigma, \text{com}, \text{peer}, \text{mes})$ and $(E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \prec'_\sigma, \text{com}, \text{peer}, \text{mes})$ Associated with the Distributed Execution in Figure 2.1. A path from e_1 to e_2 in the first run means $e_1 \prec_\sigma e_2$ (resp. $e_1 \prec'_\sigma e_2$ in the second run). Events that happen in a different order in each run are circled.

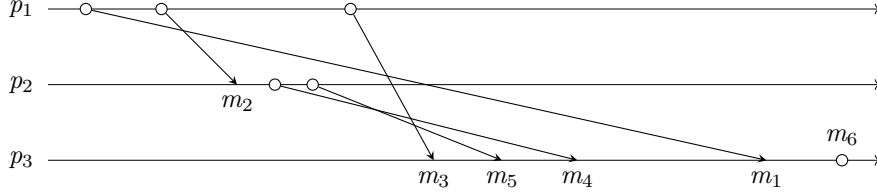


Figure 2.3: A Space-Time Diagram Providing an Alternative Representation of the Second Run in Figure 2.2 from older events on the left towards newer events on the right. Each horizontal line is a peer. An arrow corresponds to the transmission of a message from one peer to another. The two ends of an arrow are the associated send and receive events.

Theorem 4 (Causal and Total Orderings are Locally Equivalent). *Given a run $(E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \prec_\sigma, \text{com}, \text{peer}, \text{mes})$, \prec_c and \prec_σ are locally the same:*

$$\forall e_1, e_2 \in E : \text{peer}(e_1) = \text{peer}(e_2) \Rightarrow (e_1 \prec_c e_2) \Leftrightarrow (e_1 \prec_\sigma e_2)$$

Proof. By definition 1, events occurring on the same peer are totally ordered by \prec_c . By definition 2, \prec_σ is a total order that extends \prec_c . Therefore, they are locally the same. \square

Figure 2.2 provides two linear extensions of the distributed execution depicted in Figure 2.1. In the distributed execution, there is no causal dependency between the send event of m_3 and the send event of m_4 . This is also the case for the send events of m_3 and m_5 . This is why there are two runs in which those events happen in a different order. However, the send events of m_4 and m_5 are causally related (they both happen on p_2): therefore the ordering is always preserved in the runs that are derived from the distributed execution. This is also the case for the send and receive events of m_3 . Other runs derived from the distributed execution involve the interleaving of causally unrelated events. Here, for any event that happens on p_2 , it is independent from the send event or receive event of m_3 (two unrelated branches in the partially ordered set). Figure 2.3 is a space-time diagram that describes the second run. It is a convenient representation of the total ordering of events in the run that also emphasises the local total ordering of events on each peer and the transmission of messages. In addition, it is easy to read the layout of the inherent distributed execution: causal precedence corresponds to paths (on a peer or through transmission of messages) in the diagram. For instance, the send event of m_2 precedes the receive event of m_5 . On the diagram there are two causal paths of independent events between them:

- from p_1 to p_2 then p_3 with the transmission of m_2 followed by the transmission of m_5 ;
- from p_1 to p_3 with the transmission of m_3 .

Lemma 5. Given $(E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \text{com}, \text{peer}, \text{mes})$ a distributed execution, \prec_c is the smallest relation on E such that:

$$\forall e_1, e_2 \in E : e_1 \prec_c e_2 \Leftrightarrow \left(\begin{array}{c} e_1 \prec_c^{\text{elem}} e_2 \\ \vee \exists e_3 \in E : \left(\begin{array}{c} e_1 \neq e_3 \\ \wedge e_1 \prec_c^{\text{elem}} e_3 \\ \wedge e_3 \prec_c e_2 \end{array} \right) \end{array} \right)$$

where:

$$e_1 \prec_c^{\text{elem}} e_2 \triangleq \left(\begin{array}{c} (e_1 \prec_c e_2 \wedge \text{peer}(e_1) = \text{peer}(e_2)) \\ \vee \left(\begin{array}{c} \text{com}(e_1) = \text{Send} \\ \wedge \text{com}(e_2) = \text{Receive} \\ \wedge \text{mes}(e_1) = \text{mes}(e_2) \end{array} \right) \end{array} \right)$$

Proof. This is the transformation of “transitivity n-n” to “transitivity 1-n”, and the two definitions are proven equivalent in the Coq Standard Library (module **Relations**). \square

Theorem 6. Given $(E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \text{com}, \text{peer}, \text{mes})$ a distributed execution, two events that are causally dependent and happen on two different peers are causally linked by the transmission of a message.

$$\forall e_1, e_2 \in E : \left(\begin{array}{c} \text{peer}(e_1) \neq \text{peer}(e_2) \\ \wedge e_1 \prec_c e_2 \end{array} \right) \Rightarrow \exists e_s, e_r \in E : \left(\begin{array}{c} \text{com}(e_s) = \text{Send} \\ \wedge \text{com}(e_r) = \text{Receive} \\ \wedge \text{mes}(e_s) = \text{mes}(e_r) \\ \wedge \text{peer}(e_s) = \text{peer}(e_1) \\ \wedge e_1 \prec_c e_s \\ \wedge e_r \prec_c e_2 \end{array} \right)$$

Proof. Let $(E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \text{com}, \text{peer}, \text{mes})$ a distributed execution and $e_1, e_2 \in E$ such that

$$e_1 \prec_c e_2 \text{ and } \text{peer}(e_1) \neq \text{peer}(e_2). \text{ Let us prove that } \exists e_s, e_r \in E : \left(\begin{array}{c} \text{com}(e_s) = \text{Send} \\ \wedge \text{com}(e_r) = \text{Receive} \\ \wedge \text{mes}(e_s) = \text{mes}(e_r) \\ \wedge \text{peer}(e_s) = \text{peer}(e_1) \\ \wedge e_1 \prec_c e_s \\ \wedge e_r \prec_c e_2 \end{array} \right)$$

by induction on the principle underlying Lemma 5.

1. Case $e_1 \prec_c^{\text{elem}} e_2$.

(a) Case $e_1 \prec_c e_2 \wedge \text{peer}(e_1) = \text{peer}(e_2)$.

Impossible by hypothesis $\text{peer}(e_1) \neq \text{peer}(e_2)$.

(b) Case $\text{com}(e_1) = \text{Send} \wedge \text{com}(e_2) = \text{Receive} \wedge \text{mes}(e_1) = \text{mes}(e_2)$.

QED with $e_s \leftarrow e_1$ and $e_r \leftarrow e_2$ because \prec_c is reflexive.

2. Case $\exists e_3 \in E : e_1 \neq e_3 \wedge e_1 \prec_c^{\text{elem}} e_3 \wedge e_3 \prec_c e_2$.

(a) Case $e_1 \prec_c e_3 \wedge \text{peer}(e_1) = \text{peer}(e_3)$.

By the induction hypothesis $\exists e_s, e_r \in E : \left(\begin{array}{c} \text{com}(e_s) = \text{Send} \\ \wedge \text{com}(e_r) = \text{Receive} \\ \wedge \text{mes}(e_s) = \text{mes}(e_r) \\ \wedge \text{peer}(e_s) = \text{peer}(e_3) \\ \wedge e_3 \prec_c e_s \\ \wedge e_r \prec_c e_2 \end{array} \right).$

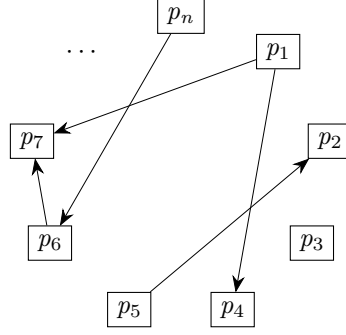


Figure 2.4: Example of Point-to-Point Communication. An arrow accounts for the transmission of a message. Each message has got one sender and one receiver.

Therefore, $\text{peer}(e_1) = \text{peer}(e_s)$ and $e_1 \prec_c e_s$ by transitivity of \prec_c .

QED with $e_s \leftarrow e_s$ and $e_r \leftarrow e_r$.

- (b) Case $\text{com}(e_1) = \text{Send} \wedge \text{com}(e_3) = \text{Receive} \wedge \text{mes}(e_1) = \text{mes}(e_3)$.

QED with $e_s \leftarrow e_1$ and $e_r \leftarrow e_3$.

□

2.2 Point-to-Point Communication

In point-to-point (or one-to-one) communication, a message is received by at most one peer. A given message is sent by a peer and may be received by another. Unless otherwise specified, this work assumes point-to-point communication. Figure 2.4 illustrates point-to-point communication and the transmission of messages between one sender and one receiver.

Definition 7 (Point-to-Point Distributed Execution and Run). *A point-to-point distributed execution $(E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \text{com}, \text{peer}, \text{mes})$ is a distributed execution where no message is sent or received more than once.*

$$\forall e, e' \in E : \left(\begin{array}{l} \text{com}(e) = \text{com}(e') = \text{Send} \\ \vee \text{com}(e) = \text{com}(e') = \text{Receive} \end{array} \right) \wedge \text{mes}(e) = \text{mes}(e') \Rightarrow e = e'$$

The set of point-to-point distributed executions on $(\mathcal{P}, \mathcal{M})$ is denoted Exec^{P2P} . The set of runs that extend point-to-point distributed executions is $\text{Run}^{\text{P2P}} \triangleq \text{Runs}(\text{Exec}^{\text{P2P}})$.

The distributed execution depicted in Figure 2.1 is a point-to-point distributed execution. There is indeed at most one send event and one receive event for a given message. Note that the definition allows sent messages to never be received (a message m such that $\exists e \in E : \text{com}(e) = \text{Send} \wedge \text{mes}(e) = m \wedge \neg \exists e' \in E : (\text{com}(e') = \text{Receive} \wedge \text{mes}(e') = m)$). This can be interpreted as the loss of the message, and is indistinguishable from a message staying forever in transit. In the example, m_6 is such a message.

2.3 Group Communication

In group (or one-to-many) communication, a message is sent by a peer and may be received by several others. Figure 2.5 illustrates multicast communication: in this example p_1 sends a

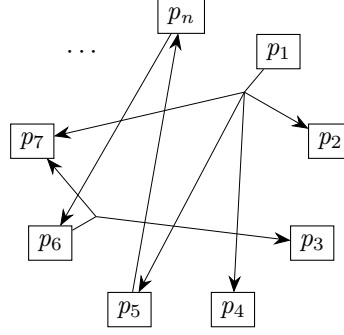


Figure 2.5: Example of Multicast Communication. An arrow, from one peer to another, accounts for the transmission of a message. A message has got only one sender but some messages can have more than one receiver although they are sent only once.

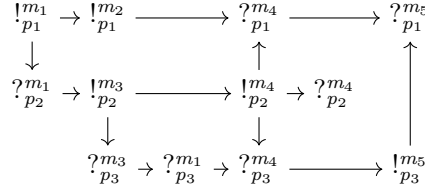


Figure 2.6: Example of a Multicast Distributed Execution. A send event of a message m by a peer p is denoted $!^m_p$. A receive event of a message m on a peer p is denoted $?^m_p$. A path from e_1 to e_2 means $e_1 \prec_c e_2$.

message that is received by p_2 , p_4 , and p_7 . This does not mean that the message is sent three times and received by a different peer each time. It means the message is sent once but received by those three peers. Point-to-point communication is a specific case of multicast communication: for instance, in the diagram p_5 sends a message that is received by p_n only.

Definition 8 (Multicast Distributed Execution and Run). *A multicast distributed execution $(E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \text{com}, \text{peer}, \text{mes})$ is a distributed execution where no message is sent more than once and where no message is received more than once on the same peer.*

$$\forall e, e' \in E : \text{com}(e) = \text{com}(e') = \text{Send} \wedge \text{mes}(e) = \text{mes}(e') \Rightarrow e = e'$$

$$\forall e, e' \in E : \text{com}(e) = \text{com}(e') = \text{Receive} \wedge \text{mes}(e) = \text{mes}(e') \wedge \text{peer}(e) = \text{peer}(e') \Rightarrow e = e'$$

The set of multicast distributed executions is denoted Exec . The set of runs that extend multicast distributed executions is $\text{Run} \triangleq \text{Runs}(\text{Exec})$.

Figure 2.6 depicts an example of a multicast distributed execution. In a multicast distributed execution, as in point-to-point distributed executions, a message can only be sent once. It can be received once (e.g. m_3 and m_5), not received at all (e.g. m_2), or received several times. Message m_4 is received exactly once per peer, including the sending peer p_2 : there cannot be more receive events of m_4 in a multicast distributed execution that involves three peers. Here, message m_1 is received by p_2 and p_3 but it is not received by p_1 which is valid in a multicast distributed execution.

2.4 Asynchronous Communication Models

There is a great variety of ordering properties that can specify the communication. Applicative orderings, such as message priorities, are not considered as our goal is to study the communication models without taking any specific application into account.

We present seven asynchronous communication models. There are four variants of FIFO communication, according to the peers involved: *FIFO 1-1* coordinates one sender with one receiver, *FIFO n-1* coordinates all the senders of a unique receiver, *FIFO 1-n* coordinates one sender with all its destinations, and *FIFO n-n* coordinates all the senders with all the receivers. Additionally, there are causal communication *Causal*, pseudo-synchronous communication *RSC*, and *Fully Asynchronous* communication.

Some communication model (*Fully Asynchronous*, *FIFO 1-1*, and *Causal*) consist in ordering events that are causally dependent. They are characterised by a subset of point-to-point or multicast distributed executions.

Some communication models (*FIFO 1-n*, *FIFO n-1*, *FIFO n-n*, *RSC*) involve ordering events that occur on different peers without any causal dependency. Such models need to be characterised by a subset of point-to-point or multicast runs: their specifications imply absolute time and make use of the total ordering of events that extends causality. The peers need to share knowledge in order to account for the total ordering. In distributed implementations of these models, the transmission of messages is the only way to exchange information. Additional messages have to be dedicated to the realisation of the ordering policy.

2.4.1 Fully Asynchronous Communication

No order on message delivery is imposed. Messages can overtake others or be arbitrarily delayed. The implementation is usually modeled by a bag (or a set if messages are unique). Any point-to-point or multicast distributed execution is a valid *Fully Asynchronous* distributed execution.

Definition 9 (*Fully Asynchronous Distributed Execution*). *The set of Fully Asynchronous distributed executions is Exec . The set of point-to-point Fully Asynchronous distributed executions is Exec^{P2P} . The set of Fully Asynchronous runs is Run . The set of point-to-point Fully Asynchronous runs is Run^{P2P} .*

2.4.2 FIFO 1-1 Communication

Messages transmitted between a given couple of sending peer and receiving peer are delivered in the sending order. Messages transmitted from, or to, a different peer are however independently delivered. More precisely, if a peer sends a message m_1 and later a message m_2 , and these two messages are received by a same peer, then m_2 must be received after m_1 .

Definition 10 (*FIFO 1-1 Distributed Execution and Run*). *The set Exec_{11} of FIFO 1-1 dis-*

tributed executions is:

$$\text{Exec}_{11} \triangleq \left\{ (E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \text{com}, \text{peer}, \text{mes}) \in \text{Exec} \mid \begin{array}{l} \forall es_1, es_2, er_1, er_2 \in E : \\ \left(\begin{array}{l} \text{com}(es_1) = \text{Send} \\ \wedge \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{Receive} \\ \wedge \text{com}(er_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) \\ \wedge \text{peer}(es_1) = \text{peer}(es_2) \\ \wedge \text{peer}(er_1) = \text{peer}(er_2) \\ \wedge es_1 \neq es_2 \\ \wedge es_1 \prec_c es_2 \end{array} \right) \\ \Rightarrow er_1 \prec_c er_2 \end{array} \right\}$$

The set of point-to-point FIFO 1-1 distributed executions is $\text{Exec}_{11}^{\text{P2P}} \triangleq \text{Exec}^{\text{P2P}} \cap \text{Exec}_{11}$. The sets of FIFO 1-1 runs and point-to-point runs are $\text{Run}_{11} \triangleq \text{Runs}(\text{Exec}_{11})$ and $\text{Run}_{11}^{\text{P2P}} \triangleq \text{Run}^{\text{P2P}} \cap \text{Run}_{11}$.

The definition states that a local ordering of the send events implies a local ordering of associated receive events. It orders the communication between one peer (1) and another (1) in a first in first out (FIFO) manner, hence the name *FIFO 1-1*. As a result, the model could be easily implemented with a simple queue between each pair of peer. When a peer p_1 sends a message m to peer p_2 , m is appended to a queue $q(p_1, p_2)$. When p_2 is to receive a message from p_3 , it retrieves the first message of the queue $q(p_3, p_2)$. This is illustrated in Figure 2.7.

The distributed execution in Figure 2.8a is *FIFO 1-1* because m_1 is sent before m_2 , both on peer p_1 and they are both received on p_2 in that order. In Figure 2.8b, m_2 is received before m_1 , thus the distributed execution is not *FIFO 1-1*. This is also the case in Figure 2.8c but this time, m_1 and m_2 are received on different peers so the distributed execution is *FIFO 1-1* anyway. Similarly, m_3 and m_1 are received on the same peer p_3 in the reverse order of their emission but the send events occur on different peers. Figures 2.9a, 2.9b, and 2.9c depict possible runs based on these distributed executions.

2.4.3 Causal: Causally Ordered Communication

Messages are delivered according to the causality of their send [Lam78]. More precisely, if a message m_1 is causally sent before a message m_2 , then a given peer cannot receive m_2 before m_1 .

Definition 11 (*Causal Distributed Execution and Run*). The set Exec_c of Causal distributed executions is:

$$\text{Exec}_c \triangleq \left\{ (E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \text{com}, \text{peer}, \text{mes}) \in \text{Exec} \mid \begin{array}{l} \forall es_1, es_2, er_1, er_2 \in E : \\ \left(\begin{array}{l} \text{com}(es_1) = \text{Send} \\ \wedge \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{Receive} \\ \wedge \text{com}(er_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) \\ \wedge \text{peer}(er_1) = \text{peer}(er_2) \\ \wedge es_1 \neq es_2 \\ \wedge es_1 \prec_c es_2 \end{array} \right) \\ \Rightarrow er_1 \prec_c er_2 \end{array} \right\}$$

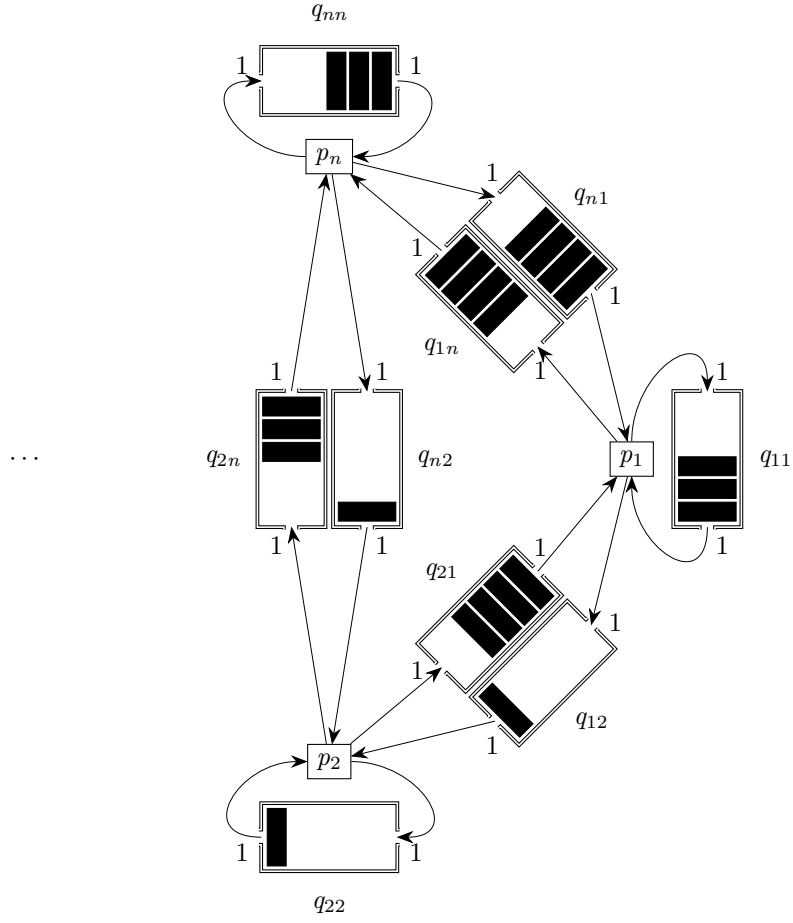


Figure 2.7: Illustration of the *FIFO 1-1* Ordering. An arrow departing from a peer towards a queue means a message is sent and inserted in the queue. An arrow departing from a queue towards a peer means the reception of a message retrieved from the queue.

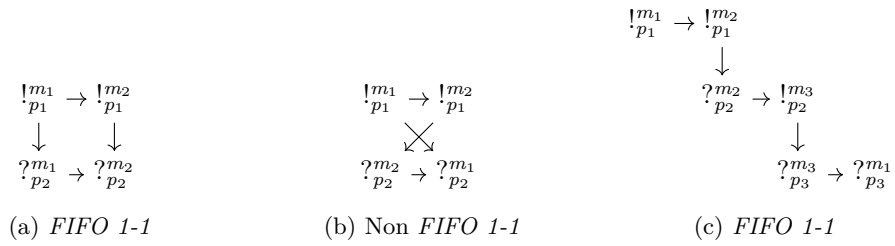


Figure 2.8: *FIFO 1-1* Ordering in Example Distributed Executions

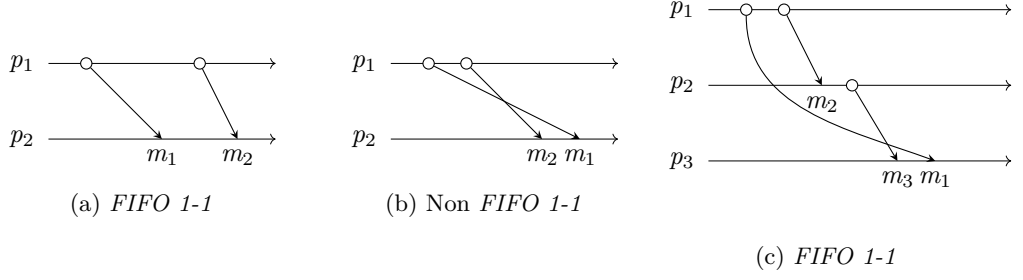


Figure 2.9: *FIFO 1-1* Ordering in Example Runs. The underlying distributed executions are depicted in Figures 2.8a, 2.8b, and 2.8c.

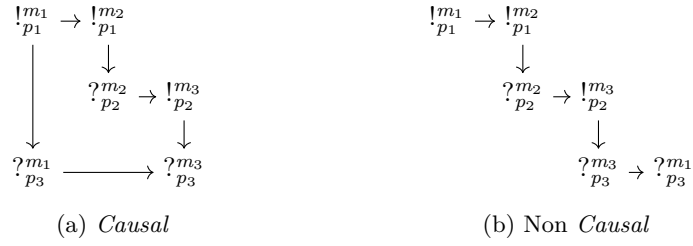


Figure 2.10: *Causal* Ordering in Example Distributed Executions

The set of point-to-point *Causal* distributed executions is $\text{Exec}_c^{\text{P2P}} \triangleq \text{Exec}^{\text{P2P}} \cap \text{Exec}_c$. The sets of *Causal* runs and point-to-point runs are $\text{Run}_c \triangleq \text{Runs}(\text{Exec}_c)$ and $\text{Run}_c^{\text{P2P}} \triangleq \text{Run}^{\text{P2P}} \cap \text{Run}_c$.

The distributed execution in Figure 2.10b, which is *FIFO 1-1* as seen previously, is not *Causal* because m_3 and m_1 are both received on the same peer p_3 in that order but the send event of m_1 actually precedes the send event of m_3 . However, in Figure 2.10a, on peer p_3 , the receive events occur in the same order so the distributed execution is *Causal*. Figures 2.11a and 2.11b depict possible runs based on these distributed executions.

An implementation of this model requires the sharing of the causality relation, using causal histories [SM94, KS98] or logical vector/matrix clocks [RST91, CDK94, PRS97, Ray13].

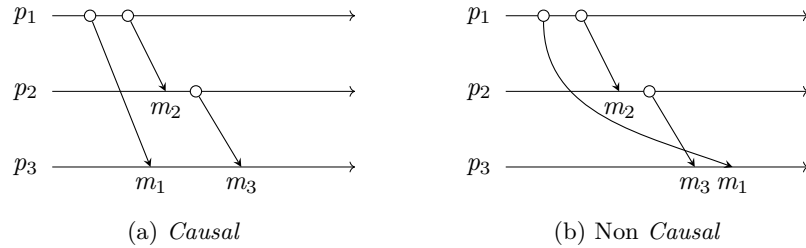


Figure 2.11: *Causal* Ordering in Example Runs. The underlying distributed executions are depicted in Figures 2.10a and 2.10b.

2.4.4 *FIFO n-1* Communication

Messages received on the same peer are received in their sending order. Even if the send events are independent, the delivery order is their sending order in absolute time. A send event is implicitly and globally ordered with regard to all other emissions toward the same peer. This means that if a peer p consumes m_1 (sent by a peer p_1) and later m_2 (sent by peer p_2), then p *knows* that the sending on peer p_1 occurs before the sending on peer p_2 in the total run order, even if there is no causal dependency between the two emissions.

Definition 12 (*FIFO n-1 Run*). The set Run_{n1} of *FIFO n-1* runs is:

$$\text{Run}_{n1} \triangleq \left\{ (E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \prec_\sigma, \text{com}, \text{peer}, \text{mes}) \in \text{Run} \mid \begin{array}{l} \forall es_1, es_2, er_1, er_2 \in E : \\ \left(\begin{array}{l} \text{com}(es_1) = \text{Send} \\ \wedge \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{Receive} \\ \wedge \text{com}(er_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) \\ \wedge \text{peer}(er_1) = \text{peer}(er_2) \\ \wedge es_1 \neq es_2 \\ \wedge es_1 \prec_\sigma es_2 \end{array} \right) \\ \Rightarrow er_1 \prec_c er_2 \end{array} \right\}$$

The set of point-to-point *FIFO n-1* runs is $\text{Run}_{n1}^{\text{P2P}} \triangleq \text{Run}^{\text{P2P}} \cap \text{Run}_{n1}$.

Note that, in the previous definition, $er_1 \prec_c er_2$ is substitutable with $er_1 \prec_\sigma er_2$ because er_1 and er_2 happen on the same peer (Theorem 4).

The definition states that a global ordering of the send events implies a local ordering of associated receive events. It orders the communication between all the n peers and a given (1) receiver in a first in first out (FIFO) manner, hence the name *FIFO n-1*. An implementation of this model requires a shared real-time clock [CF99] or a global agreement on event order [DSU04, Ray10]. For example, each peer has a unique input queue, a mailbox, in which messages are deposited instantly by senders, without blocking. When a peer p_1 sends a message m to peer p_2 , m is appended to the queue $q(p_2)$. When p_2 is to receive a message from p_1 , p_3 , or any other peer, it retrieves the first message of the queue $q(p_2)$. Thus, on a given peer, the messages are received in their absolute sending order. Figure 2.12 illustrates this.

The run in Figure 2.13a is *FIFO n-1* because m_1 is sent before m_2 , and the associated receptions that occur on the same peer p_3 happen in the same order. In Figure 2.13b however, they happen in the reverse order: the run is not *FIFO n-1*. Information about the sending peers is here irrelevant.

This model is used for instance in [BBO12, OSB13] as an abstraction of asynchronous communication. This model is often confused with the previously described *FIFO 1-1* model.

2.4.5 *FIFO 1-n* Communication

Messages from a same peer are delivered in their sending order. This model is the dual of *FIFO n-1*: it induces another global order, this time on the receivers (one for each sender).

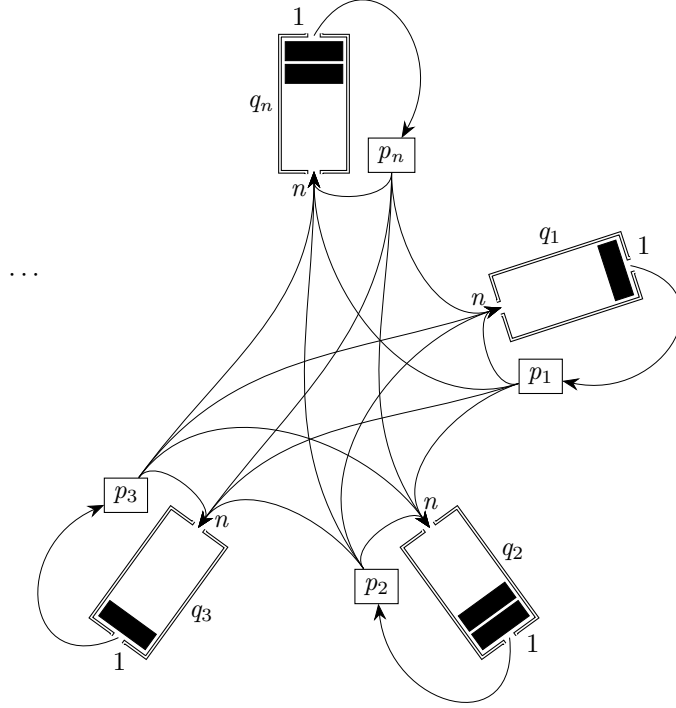


Figure 2.12: Illustration of the *FIFO* $n-1$ Ordering. A queue per peer serves as an inbox and provides *FIFO* ordering between the n peers of the system and the one (1) peer associated the this inbox. An arrow departing from a peer towards a queue means a message is sent and inserted in the queue. An arrow departing from a queue towards a peer means the reception of a message retrieved from the queue.

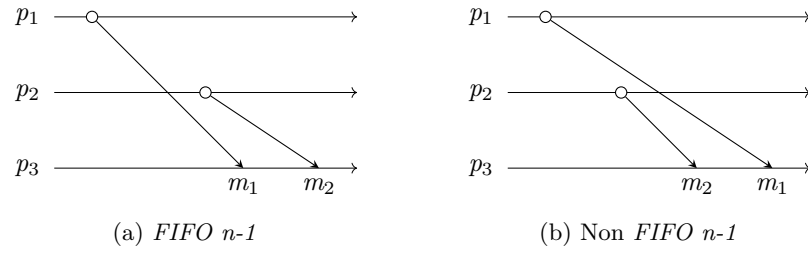


Figure 2.13: *FIFO* $n-1$ Ordering in Example Runs

Definition 13 (*FIFO 1-n Run*). The set Run_{1n} of *FIFO 1-n runs* is:

$$\text{Run}_{1n} \triangleq \left\{ (E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \prec_\sigma, \text{com}, \text{peer}, \text{mes}) \in \text{Run} \mid \begin{array}{l} \forall es_1, es_2, er_1, er_2 \in E : \\ \left(\begin{array}{l} \text{com}(es_1) = \text{Send} \\ \wedge \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{Receive} \\ \wedge \text{com}(er_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) \\ \wedge \text{peer}(es_1) = \text{peer}(es_2) \\ \wedge es_1 \neq es_2 \\ \wedge es_1 \prec_c es_2 \end{array} \right) \\ \Rightarrow er_1 \prec_\sigma er_2 \end{array} \right\}$$

The set of point-to-point *FIFO 1-n runs* is $\text{Run}_{1n}^{\text{P2P}} \triangleq \text{Run}^{\text{P2P}} \cap \text{Run}_{1n}$.

Note that, in the previous definition, $es_1 \prec_c es_2$ is substitutable with $es_1 \prec_\sigma es_2$ because es_1 and es_2 happen on the same peer (Theorem 4). The definition states that a local ordering of the send events implies a global ordering (total order on the run) of associated receive events. It orders the communication between a given sender (1) and all the n peers in a first in first out (FIFO) manner, hence the name *FIFO 1-n*. In a possible implementation, each peer has a unique queue, an outbox, where sent messages are put. Destination peers fetch messages instantly from this queue and acknowledge their reception. When a peer p_1 sends a message m to peer p_2 , p_3 , or any other peer, m is appended to the queue $q(p_1)$. When p_2 , p_3 , or any other peer is to receive a message from p_1 , it retrieves the first message of the queue $q(p_1)$. Thus, all the messages sent by a given peer are received in their sending order, wherever they are received.

The run in Figure 2.15a is *FIFO 1-n*: m_1 is sent before m_2 , both by peer p_1 , and the associated receptions occur in this order. Although the receptions occur on different peers, if the receive event of m_2 occurred before the receive event of m_1 , the run would not be *FIFO 1-n* as in Figure 2.15b.

In point-to-point communication, an alternative definition of *FIFO 1-n* only involves a causal dependency on the send events.

Theorem 14 (Alternative definition of point-to-point *FIFO 1-n Runs*). $\text{Run}_{1n}^{\text{P2P}} = \text{Run}_{1n}^{\text{P2P}'}$ where:

$$\text{Run}_{1n}^{\text{P2P}'} \triangleq \left\{ (E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \prec_\sigma, \text{com}, \text{peer}, \text{mes}) \in \text{Run} \mid \begin{array}{l} \forall es_1, es_2, er_1, er_2 \in E : \\ \left(\begin{array}{l} \text{com}(es_1) = \text{Send} \\ \wedge \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{Receive} \\ \wedge \text{com}(er_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) \\ \wedge es_1 \neq es_2 \\ \wedge es_1 \prec_c es_2 \end{array} \right) \\ \Rightarrow er_1 \prec_\sigma er_2 \end{array} \right\}$$

Proof.

1. Proof of $\text{Run}_{1n}^{\text{P2P}'} \subseteq \text{Run}_{1n}^{\text{P2P}}$

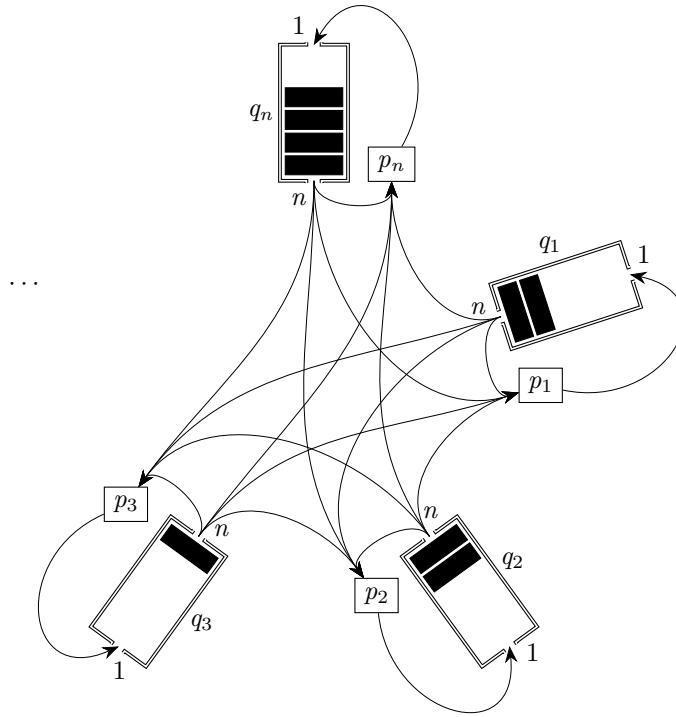


Figure 2.14: Illustration of the *FIFO 1-n* Ordering. A queue per peer serves as an outbox and provides FIFO ordering between the one (1) sender associated with this outbox and all the n peers of the system. An arrow departing from a peer towards a queue means the send of a message and its insertion in the queue. An arrow departing from a queue towards a peer means the reception of a message retrieved from the queue.

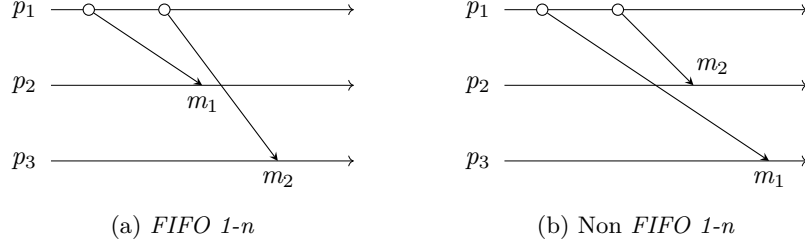


Figure 2.15: *FIFO 1-n* Ordering in Example Runs

Let $(E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \prec_\sigma, \text{com}, \text{peer}, \text{mes}) \in \text{Run}_{1n}^{\text{P2P}'}$. $\forall e_1, e_2 \in E : \text{peer}(e_1) = \text{peer}(e_2) \wedge e_1 \prec_c e_2 \Rightarrow e_1 \prec_\sigma e_2$. Hence $\text{Run}_{1n}^{\text{P2P}'} \subseteq \text{Run}_{1n}^{\text{P2P}}$.

2. Proof of $\text{Run}_{1n}^{\text{P2P}} \subseteq \text{Run}_{1n}^{\text{P2P}'}$

Assume $r \triangleq (E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \prec_\sigma, \text{com}, \text{peer}, \text{mes}) \in \text{Run}_{1n}^{\text{P2P}}$ and $r \notin \text{Run}_{1n}^{\text{P2P}'}$.

$$\exists es_1, es_2, er_1, er_2 \in E : \left(\begin{array}{l} \text{com}(es_1) = \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{com}(er_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) \\ \wedge es_1 \neq es_2 \\ \wedge es_1 \prec_c es_2 \end{array} \right), \text{ but } er_2 \prec_\sigma er_1.$$

(a) Case $\text{peer}(es_1) = \text{peer}(es_2)$

Contradiction since $r \in \text{Run}_{1n}^{\text{P2P}}$, then $er_1 \prec_\sigma er_2$ and $r \in \text{Run}_{1n}^{\text{P2P}'}$.

(b) Case $\text{peer}(es_1) \neq \text{peer}(es_2)$

by Theorem 6, $\exists es, er \in E : \text{com}(es) = \text{Send} \wedge \text{com}(er) = \text{Receive} \wedge \text{mes}(es) = \text{mes}(er) \wedge \text{peer}(es_1) = \text{peer}(es) \wedge es_1 \prec_c es \wedge er \prec_c es_2$

i. Case $es = es_1$

$er = er_1$ because $r \in \text{Run}^{\text{P2P}}$ (at most one reception in point-to-point communication).

$er_1 \prec_c es_2$ because $er \prec_c es_2$ and $er = er_1$.

$es_2 \prec_c er_2$ by transmission of a message.

Contradiction. $er_1 \prec_c er_2$ by transitivity.

ii. Case $es \neq es_1$

$es \prec_c er$ and $es_2 \prec_c er_2$ by transmission of messages

$er \prec_\sigma er_2$ by transitivity of \prec_c and \prec_σ extends \prec_c

Contradiction. $r \notin \text{Run}_{1n}^{\text{P2P}}$ because $er \prec_\sigma er_1$ (since $er_2 \prec_c er_1$), $es \neq es_1$, but $es_1 \prec_c es$.

□

2.4.6 *FIFO n-n* Communication

Messages are globally ordered and are delivered in their emission order. This model can be based on a shared centralised object (e.g. a unique queue). Usually, it is used as a first step to move away from the synchronous communication model, by splitting send and receive events.

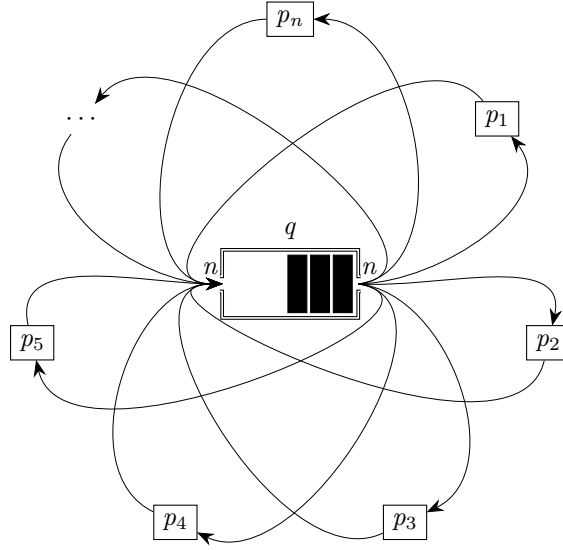


Figure 2.16: Illustration of the *FIFO n-n* Ordering. A unique centralised object provides FIFO ordering between all the n peers. An arrow departing from a peer towards a queue means the send of a message and its insertion in the queue. An arrow departing from a queue towards a peer means the reception of a message retrieved from the queue.

Definition 15 (*FIFO n-n Run*). The set Run_{nn} of *FIFO n-n* runs is:

$$\text{Run}_{nn} \triangleq \left\{ (E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \prec_\sigma, \text{com}, \text{peer}, \text{mes}) \in \text{Run} \mid \begin{array}{l} \forall es_1, es_2, er_1, er_2 \in E : \\ \left(\begin{array}{l} \text{com}(es_1) = \text{Send} \\ \wedge \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{Receive} \\ \wedge \text{com}(er_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) \\ \wedge es_1 \neq es_2 \\ \wedge es_1 \prec_\sigma es_2 \\ \Rightarrow er_1 \prec_\sigma er_2 \end{array} \right) \end{array} \right\}$$

The set of point-to-point *FIFO n-n* runs is $\text{Run}_{nn}^{\text{P2P}} \triangleq \text{Run}^{\text{P2P}} \cap \text{Run}_{nn}$.

The definition states that a global ordering of the send events implies a global ordering of associated receive events. It orders the communication between all the peers (n) and all the peers (n) in a first in first out (FIFO) manner, hence the name *FIFO n-n*. As a result, the model can be easily described with a simple global queue that contains all the messages in transit. When a peer p_1 sends a message m to peer p_2 , m is appended to that queue q . When p_2 is to receive a message it retrieves the first message of the queue q . This is illustrated by Figure 2.16.

The run in Figure 2.17b is not *FIFO n-n* because m_1 is sent before m_2 but it is received after, even though the involved peers are all different. The messages are however received in the same order in the *FIFO n-n* run from Figure 2.17a.

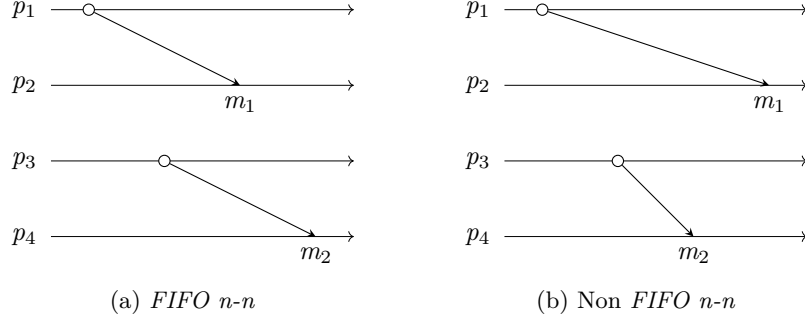


Figure 2.17: *FIFO n-n* Ordering in Example Runs

2.4.7 *RSC*: Realisable with Synchronous Communication

A point-to-point run is *realizable with synchronous communication* if each send event is immediately followed by its corresponding receive event [CBMT96, KS11]. If the couple of send and receive events is viewed atomically, this corresponds to a synchronous communication execution. In multicast communication, this means that a send event is immediately followed by the corresponding receive events or internal events. After another message is sent, no more receptions of the previous message can happen in the run.

Definition 16 (*RSC Run*). *The set Run_{RSC} of *RSC* runs is:*

$$\text{Run}_{RSC} \triangleq \left\{ \left(E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \prec_\sigma, \right) \in \text{Run} \mid \begin{array}{l} \forall es, er, e \in E : \\ \left(\begin{array}{l} \text{com}(es) = \text{Send} \\ \wedge \text{com}(er) = \text{Receive} \\ \wedge \text{mes}(es) = \text{mes}(er) \\ \wedge es \prec_\sigma e \prec_\sigma er \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} e = es \\ \vee \text{com}(e) = \text{Internal} \\ \vee \left(\begin{array}{l} \text{com}(e) = \text{Receive} \\ \wedge \text{mes}(e) = \text{mes}(er) \end{array} \right) \end{array} \right) \end{array} \right\}$$

The set of point-to-point *RSC* runs is $\text{Run}_{RSC}^{\text{P2P}} \triangleq \text{Run}^{\text{P2P}} \cap \text{Run}_{RSC}$.

The definition states that between a send event and a receive event of a given message, there can only be other receive events of that message or internal events.

The run in Figure 2.18b is not *RSC* because, although the two messages are exchanged by different couples of peers, m_2 is sent before m_1 is received. Two messages must not be in transit at the same time. However, the run in Figure 2.18a is *RSC* because m_1 is received first.

2.4.8 Summary of the Asynchronous Communication Models

The asynchronous communication models that have been presented, with the exception of the *Fully Asynchronous* and *Realisable with Synchronous Communication* communication models, are characterised by two kinds of orderings among causal, local, and global.

1. An event can causally precede another when they are related in the distributed execution.
2. An event can locally precede another when they occur on the same peer.

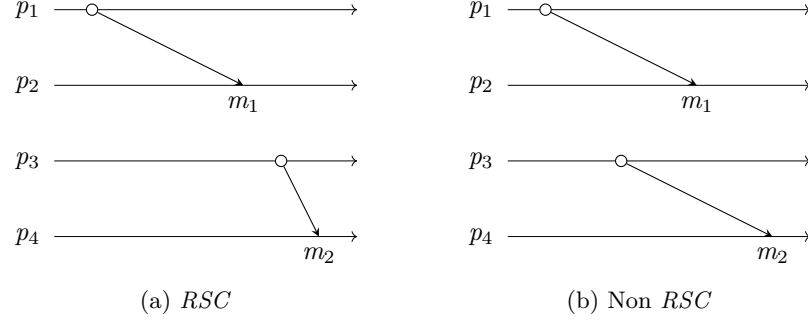


Figure 2.18: *RSC* Ordering in Example Runs

			Send		
			Causal	Local	Global
			$es_1 \prec_c es_2$	$es_1 \prec_c es_2$ \wedge $\text{peer}(es_1) = \text{peer}(es_2)$	$es_1 \prec_\sigma es_2$
Receive	Local	$er_1 \prec_c er_2$ \wedge $\text{peer}(er_1) = \text{peer}(er_2)$	<i>Causal</i>	<i>FIFO 1-1</i>	<i>FIFO n-1</i>
	Global	$er_1 \prec_\sigma er_2$	<i>FIFO 1-n</i>	<i>FIFO 1-n</i> only in point-to-point communication (Theorem 14)	<i>FIFO n-n</i>

Table 2.1: Ordering Policy Resulting from the Dependency between the Ordering of Send Events and the Ordering of Receive Events in Runs. Given a distributed execution $(E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \text{com}, \text{peer}, \text{mes})$ and two couples of send and receive events es_1, es_2, er_1 , and er_2 corresponding to the transmission of two messages ($\text{mes}(es_1) = \text{mes}(er_1)$, $\text{mes}(es_2) = \text{mes}(er_2)$, $\text{com}(es_1) = \text{com}(es_2) = \text{Send}$, and $\text{com}(er_1) = \text{com}(er_2) = \text{Receive}$), if the relation between the send events in a column implies the relation between the receive events in a row then it corresponds to the communication model in the associated cell.

3. When it comes to the total ordering on runs, an event can also globally precede another.

Table 2.1 recaps which ordering policy arises when one type of precedence between send events implies one type of precedence between receive events. In Theorem 14, the equivalence between two alternative specifications of the *FIFO 1-n* communication model has been proven: this equivalence shows up in the table and may come in handy to facilitate modelling, reasoning, proof, or implementation, depending on the context (whether the focus is on locality or causality).

2.5 Hierarchy between the Models based on the Inclusion of the Sets of Runs

There is a hierarchy between the communication models that arises from the ordering properties on the events. For instance, any communication model is stricter than the *Fully Asynchronous* communication model because the runs that are valid under the six other models are also valid under the *Fully Asynchronous* model. We base the hierarchical structure of the communication

models on the inclusion of their characteristic sets of runs. For instance, the *Causal* runs are *FIFO 1-1* runs, hence *Causal* is stricter than *FIFO 1-1*. Given a run, knowing the hierarchy between the models might allow to substitute a model with another that is stricter. This is of great interest in the study of the compatibility of communicating peers that will be detailed in Chapter 4. A thorough analysis of the hierarchy between the communication models based on refinement is presented in Chapter 6.

Theorem 17 (Hierarchy of the Seven Communication Models). *The hierarchy of the seven communication models in the general multicast case is:*

- $\text{Run}_{RSC} \subsetneq \text{Run}_{nn} \subsetneq \text{Run}_{n1} \subsetneq \text{Run}_c \subsetneq \text{Run}_{11} \subsetneq \text{Run}$
- $\text{Run}_{RSC} \subsetneq \text{Run}_{nn} \subsetneq \text{Run}_{1n} \subsetneq \text{Run}_{11} \subsetneq \text{Run}$
- Run_{n1} and Run_{1n} are not comparable
- Run_{1n} and Run_c are not comparable

In point-to-point communication, $\text{Run}_{1n}^{\text{P2P}} \subsetneq \text{Run}_c^{\text{P2P}}$:

- $\text{Run}_{RSC}^{\text{P2P}} \subsetneq \text{Run}_{nn}^{\text{P2P}} \subsetneq \text{Run}_{n1}^{\text{P2P}} \subsetneq \text{Run}_c^{\text{P2P}} \subsetneq \text{Run}_{11}^{\text{P2P}} \subsetneq \text{Run}^{\text{P2P}}$
- $\text{Run}_{RSC}^{\text{P2P}} \subsetneq \text{Run}_{nn}^{\text{P2P}} \subsetneq \text{Run}_{1n}^{\text{P2P}} \subsetneq \text{Run}_c^{\text{P2P}} \subsetneq \text{Run}_{11}^{\text{P2P}} \subsetneq \text{Run}^{\text{P2P}}$
- $\text{Run}_{n1}^{\text{P2P}}$ and $\text{Run}_{1n}^{\text{P2P}}$ are not comparable

Two Venn diagrams illustrate the hierarchy of the communication models in multicast and point-to-point communication respectively in Figure 2.19a and Figure 2.19b.

Most of the inclusions arise from the definition of the orders and are the same in multicast and point-to-point communication. The relation between *FIFO 1-n* and *Causal* is however tricky: the inclusion does not hold in multicast communication.

Let $(E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \prec_\sigma, \text{com}, \text{peer}, \text{mes}) \in \text{Run}$ and $es_1, es_2, er_1, er_2 \in E$.

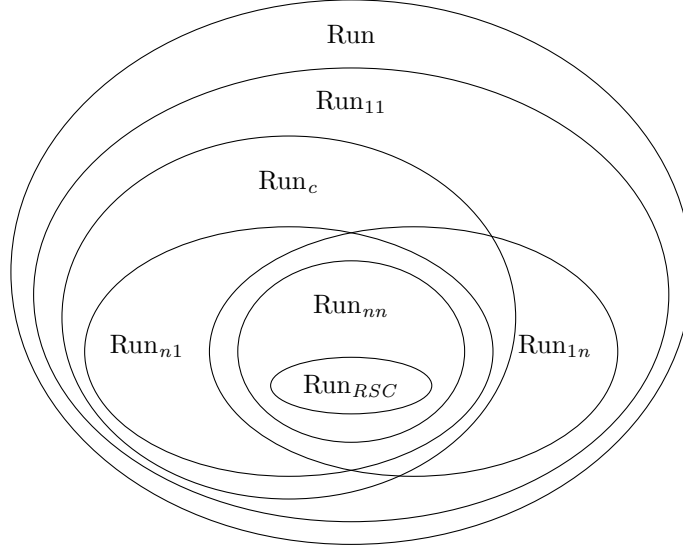
- $\text{Run}_{nn} \subseteq \text{Run}_{1n}$ and $\text{Run}_{nn}^{\text{P2P}} \subseteq \text{Run}_{1n}^{\text{P2P}}$ because

$$\left(\begin{array}{l} \text{com}(es_1) = \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{com}(er_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) \\ \wedge \text{peer}(es_1) = \text{peer}(es_2) \\ \wedge es_1 \neq es_2 \\ \wedge es_1 \prec_c es_2 \end{array} \right) \Rightarrow \left(\begin{array}{l} \text{com}(es_1) = \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{com}(er_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) \\ \wedge \text{peer}(er_1) = \text{peer}(er_2) \\ \wedge es_1 \neq es_2 \\ \wedge es_1 \prec_\sigma es_2 \end{array} \right)$$

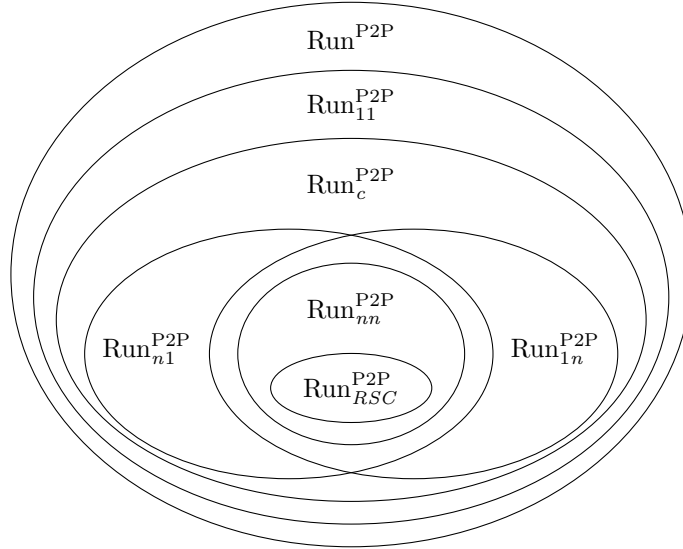
since $es_1 \prec_c es_2 \Rightarrow es_1 \prec_\sigma es_2$ by Lemma 3.

- $\text{Run}_{nn} \subseteq \text{Run}_{n1}$ and $\text{Run}_{nn}^{\text{P2P}} \subseteq \text{Run}_{n1}^{\text{P2P}}$ because

$$\left(\begin{array}{l} \text{com}(es_1) = \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{com}(er_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) \\ \wedge \text{peer}(er_1) = \text{peer}(er_2) \\ \wedge es_1 \neq es_2 \\ \wedge es_1 \prec_\sigma es_2 \end{array} \right) \Rightarrow \left(\begin{array}{l} \text{com}(es_1) = \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{com}(er_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) \\ \wedge es_1 \neq es_2 \\ \wedge es_1 \prec_\sigma es_2 \end{array} \right)$$



(a) Multicast Communication



(b) Point-to-Point Communication

Figure 2.19: Inclusion of the Sets of Runs of Communication Models for Multicast and Point-to-Point Communication

- $\text{Run}_{n1} \subseteq \text{Run}_c$ and $\text{Run}_{n1}^{\text{P2P}} \subseteq \text{Run}_c^{\text{P2P}}$ because

$$\left(\begin{array}{l} \text{com}(es_1) = \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{com}(er_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) \\ \wedge \text{peer}(er_1) = \text{peer}(er_2) \\ \wedge es_1 \neq es_2 \\ \wedge es_1 \prec_c es_2 \end{array} \right) \Rightarrow \left(\begin{array}{l} \text{com}(es_1) = \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{com}(er_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) \\ \wedge \text{peer}(er_1) = \text{peer}(er_2) \\ \wedge es_1 \neq es_2 \\ \wedge es_1 \prec_\sigma es_2 \end{array} \right)$$

since $es_1 \prec_c es_2 \Rightarrow es_1 \prec_\sigma es_2$ by Lemma 3.

- $\text{Run}_c \subseteq \text{Run}_{11}$ and $\text{Run}_c^{\text{P2P}} \subseteq \text{Run}_{11}^{\text{P2P}}$ because

$$\left(\begin{array}{l} \text{com}(es_1) = \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{com}(er_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) \\ \wedge \text{peer}(es_1) = \text{peer}(es_2) \\ \wedge \text{peer}(er_1) = \text{peer}(er_2) \\ \wedge es_1 \neq es_2 \\ \wedge es_1 \prec_c es_2 \end{array} \right) \Rightarrow \left(\begin{array}{l} \text{com}(es_1) = \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{com}(er_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) \\ \wedge \text{peer}(er_1) = \text{peer}(er_2) \\ \wedge es_1 \neq es_2 \\ \wedge es_1 \prec_c es_2 \end{array} \right)$$

- $\text{Run}_{11} \subseteq \text{Run}$ and $\text{Run}_{11}^{\text{P2P}} \subseteq \text{Run}^{\text{P2P}}$ by definition of Exec_{11} .
- $\text{Run}_{1n}^{\text{P2P}} \subseteq \text{Run}_c^{\text{P2P}}$, based on the alternate definition of *FIFO 1-n* runs from Theorem 14 because

$$\left(\begin{array}{l} \text{com}(es_1) = \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{com}(er_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) \\ \wedge \text{peer}(er_1) = \text{peer}(er_2) \\ \wedge es_1 \neq es_2 \\ \wedge es_1 \prec_c es_2 \end{array} \right) \Rightarrow \left(\begin{array}{l} \text{com}(es_1) = \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{com}(er_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) \\ \wedge es_1 \neq es_2 \\ \wedge es_1 \prec_c es_2 \end{array} \right)$$

- $\text{Run}_{RSC} \subseteq \text{Run}_{nn}$

Proof. Let $(E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \prec_\sigma, \text{com}, \text{peer}, \text{mes}) \in \text{Run}_{RSC}$.

Let $es_1, es_2, er_1, er_2 \in E$ such that

$$\left(\begin{array}{l} \text{com}(es_1) = \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{com}(er_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) \\ \wedge es_1 \neq es_2 \\ \wedge es_1 \prec_\sigma es_2 \end{array} \right)$$

$\text{mes}(er_1) \neq \text{mes}(er_2)$ because $\text{mes}(es_1) \neq \text{mes}(es_2)$ by definition of a multicast run (a message is sent at most once) since $es_1 \neq es_2$.

Assume $er_2 \prec_\sigma er_1$.

$er_2 \neq es_1$ because $\text{com}(es_1) \neq \text{com}(er_2)$.

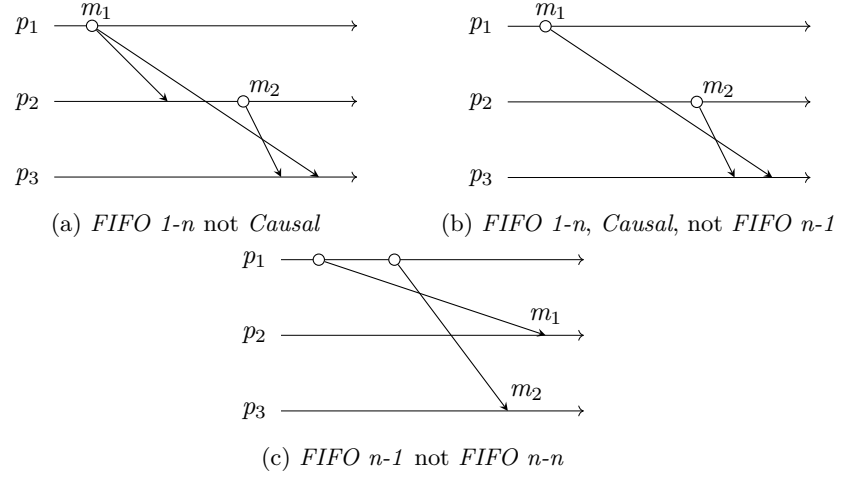


Figure 2.20: Additional Example Runs

$er_2 = es_1 \vee \text{mes}(er_1) = \text{mes}(er_2)$ by the property of Run_{RSC} with $\begin{pmatrix} es \leftarrow es_1 \\ er \leftarrow er_1 \\ e \leftarrow er_2 \end{pmatrix}$.

Contradiction. $\text{mes}(er_1) \neq \text{mes}(er_2) \wedge er_2 \neq es_1$

QED. $er_1 \prec_\sigma er_2$

□

The inclusions are strict as demonstrated by the examples referenced in Figure 2.21.

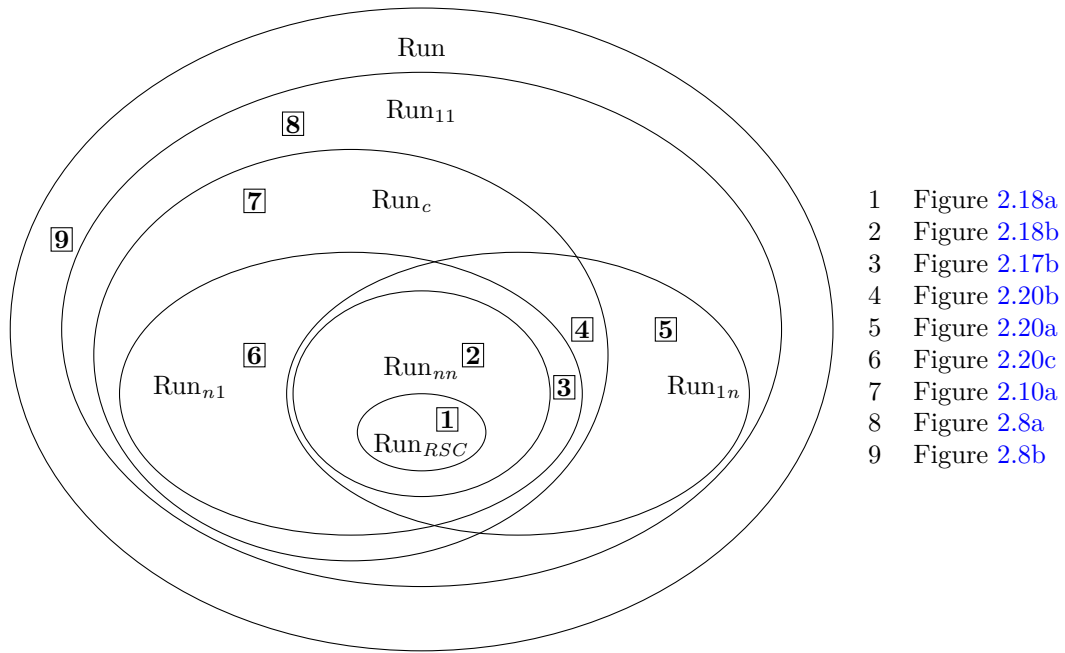


Figure 2.21: Example Runs that Prove the Strict Inclusions. Each number correspond to a figure that illustrates a distributed execution or run.

Chapter 3

State of the Art

Before getting to the heart of our contributions to the formal aspects of asynchronous communication developed in the following chapters, we provide some context and explore existing work carried out so far around the areas of distributed systems, formal specifications, and the concepts Chapter 2 has already introduced. When applicable, we draw parallels with our work or detail why existing approaches might not be adapted to the pursuit of our goals.

Approaches for the description of distributed systems are a natural starting point in this overview. The nature of the specification of systems happens to be crucial to the work developed in Chapter 4 about the compatibility of communicating peers under the different communication models. We put up a non exhaustive inventory of existing approaches and draw parallels with our work. Regarding asynchronous communication, we then contextualise the results from Chapter 2 and highlight how they help extend classic results in the domain. Eventually, we explore the motives for formal verification of distributed systems and approaches that meet those needs with an emphasis on refinement which plays a major role in our work in Chapter 6.

3.1 Description of Distributed Systems

3.1.1 Transition Systems

A classic approach we follow throughout this work consists in describing distributed systems with transition systems. Tel’s textbook [Tel00] describes a distributed system as a “collection of processes and a communication subsystem”. Each process is a transition system, and the transition system induced under asynchronous communication is built with the product of the process transition systems extended with a collection of messages in transit, and two rules for “send” and “receive”. This is similar to the framework for the verification of communicating peers we describe in Chapter 4. His formal definition considers synchronous and *Fully Asynchronous* communication. *FIFO 1-1* and *Causal* communication are mentioned but are not formalised. Tel’s goal is to describe distributed algorithms whereas our objective is to study communication models. In our work, we explicitly describe the communication models with transition systems and we compare them. We also base a framework for the verification of asynchronously communicating peers on these models.

Finite-state machines are used to represent communicating services in several verification frameworks such as in [CLB08] and, notably, in the field of web services [DOS12, BCT04, FUMK04].

Moreover, transition systems are at the heart of formal specification and verification languages

such as TLA⁺ [Lam02] (Temporal Logic of Actions), Promela [Hol04, chap. 3] (Process Meta Language), or the Event-B method [Abr10]. They allow to specify state transition systems that may describe distributed systems and asynchronous interactions. Our work rely heavily on TLA⁺ and the Event-B method. Promela models communication with FIFO message channels which does not fit our need for a less restrictive approach that encompasses the variety of asynchronous communication models.

Many other formalisms may also represent the communicating entities in distributed systems, their interactions, or their correct behaviour: process calculi, petrinets [LFS⁺11, TFZ09, Mar03], interaction diagrams, or choreographies. The remainder of this section details these formalisms.

3.1.2 I/O Automata

Input/output automata [Lyn96] provide a generic way to describe components that interact with each other thanks to input and output actions. Those actions are partitioned into tasks over which fairness properties can be defined in the same way fairness properties can be set over TLA⁺ actions. Components can either describe processes or communication channels. They can also be composed and some output actions can be made internal (hiding) in order to specify complex systems. I/O automata are said to be “input-enabled”: every input action of an automata is required to be enabled in every state, in order to avoid “the failure to specify what the component does in the face of unexpected inputs” [Lyn96, p. 203].

In our work in Chapter 4, a property called “stability with regard to interest” (Definition 20 on page 63) plays this role. It ensures the receptions of messages that might be of interest later are specified. They may result in a specified faulty state which accounts for the unexpectedness of the input.

I/O automata can model asynchronous systems in a broad sense and provide a powerful framework to describe distributed systems. However, few automatic tools have been developed to make use of I/O automata and perform modelling and property checking.

3.1.3 Message Sequence Charts

Message Sequence Charts are convenient diagrams that allow to describe the desired interactions between components that exchange messages in a system. It is an ITU standard (ITU recommendation Z.120) of the SDL (Specification and Description Language) family (ITU recommendation Z.100) that is used to describe telecommunication protocols. Message Sequence Charts characterise traces with input and output events in the distributed systems. Communication is asynchronous but the Message Sequence Charts do not assume any particular communication model whereas the peers have buffers in SDL. Thus, MSCs may be used to extend the modelling of interactions in SDL. There are two common ways to formalise the semantics of the MSC:

- With a process algebra as standardised in ITU recommendation Z.120 Annex B and presented in [MR94] by Mauw and Renier.
- With partially ordered sets of events as introduced by Alur *et al.* in [AHP96]. They consider the local ordering of events on each peer and the ordering of couples of send and receive events which corresponds exactly to our model of distributed executions in Chapter 2.

Engels *et al.* [EMR02] build on the second formalisation for their hierarchy of communication models (see Section 3.2.2). So does Longuet [Lon12] who proposes frameworks for the testing of systems and their conformance to specifications using Message Sequence Graphs (MSG). MSGs specify distributed systems with a graph structure whose nodes are MSCs. Several resulting

MSCs (that fit the notion of distributed executions in our work) can be derived. A conformance relation formalises what it means for a system to be tested against its specification. Trace inclusion of global observations of a system is an example of global conformance relation. In Chapter 4, the framework we propose allows to perform global testing; it implies a global knowledge of the system. Longuet also exposes conformance relations for local testing that may consist, for example, in isolating each peer individually and observe the interactions with the rest of the system or a set of testing peers. Eventually, the conditions for the equivalence of global and local conformance relations are established and allow to locally test if a system globally conforms to its specification.

Alur and Yannakakis propose to perform model checking of temporal properties over Message Sequence Charts in [AY99]. It relies on the formalisation of the MSCs with partial ordered sets of events [AHP96] and the automata that describe the different linear extensions. With MSC-graphs that allow to concatenate MSCs, the model checking problem with the asynchronous approach of concatenation is undecidable [AY99].

3.1.4 Choreographies and Compatibility Checking

Collaborations and choreographies are pivotal in the web services field and we can draw parallels between our work on compatibility checking in Chapter 4 and the compatibility checking of collaborations and choreographies. Collaborations describe the interactions between entities while choreographies characterise the expected behaviour resulting from these interactions. As an example, BPMN diagrams (Business Process Model and Notation), despite a partial lack of formal grounding, are a standardised approach to graphically describe such models.

Compatibility of services or software components has largely been studied, with two main goals:

- Can peers communicate and provide more complex services?
- Can one peer be replaced by another one (substitutability)?

These two notions of compatibility are different. In the first case, the peers must be complementary. In the second case however, they shall provide the same functionality: it is classically expressed by the notion of simulation (as in [ABDF08]) or the notion of trace inclusion (as in [CLB08]). In this taxonomy, we can also include different models of failure traces [GGH⁺10], where refusal sets may be used to model the receiving capabilities of the process and their preservation, thus the absence of forever pending messages. In this work, we mainly focus on the first aspect. Many approaches exist to verify behavioural compatibility of web services or software components.

Different criteria are used to represent compatibility for choreographies: deadlock freeness [DOS12, FUMK04], the absence of unspecified receptions [BZ83, DOS12], possible termination (at least one execution leads to a terminal state) [DOS12, BCT04, DWZ⁺06, LFS⁺11], certain termination (all the executions lead to a terminal state) [BCT04, BCPV04], progress (no starvation) [FUMK04], or divergence [BCPV04]. Here, divergence is a dual notion of termination that is adapted to systems that are, as most web services, expected to go on running without actually terminating. Domain application conditions are also used. In [CLB08], the behavioural compatibility of web services corresponds to a specific concept of substitutability (here trace inclusion). In [CPT01], Canal *et al.* formalises a notion of conformance between components that are specified by π -calculus processes.

Synchronous communication and the FIFO communication models are common interaction paradigms in this field. In [DOS12, BCT04, FUMK04, DWZ⁺06, BCPV04, CPT01], the communication is synchronous. *FIFO n-1* is another communication paradigm in this field [BBO12,

OSB13] that might not be clearly distinguished from the *FIFO 1-1* model encountered, for instance, in [BZ83].

In Chapter 4 and Chapter 5, we consider all the main compatibility criteria and the possibility to specify *ad hoc* properties. Furthermore, the compatibility checking is not restricted to specific communication models.

Brand and Zafropulo’s approach for unspecified receptions in [BZ83] and our property of stability with regard to interest (see Definition 20 on page 63) are two sides of the same coin. In Brand and Zafropulo’s work, if a peer can receive a given message in some state, then it must also accept this message later, in a successor state (accessible via send events). In other words, if a message can be received at a given state, its reception must also be specified at later states for a system to be correct with regard to unspecified receptions. In our work, we reverse the proposition: if a message can be received, the communication model may deliver it earlier and the system must expect this situation. This is the property of stability with regard to interest we detail in Chapter 4.

To sum up, although some works are dedicated to several compatibility criteria, all of them are dedicated to one communication model, mostly the synchronous model. None of them proposes a verification parameterised by both the compatibility criteria and multiple communication models. Moreover, only a few approaches also provide a tool to automatically check the proposed composition. One of our contribution is a unified formalisation of several communication models and compatibility criteria in a framework that checks the correctness of a composition of peers in a unified manner under any combination of the communication models. Lastly, the mechanisation of this framework returns examples of invalid runs when a compatibility criterion is violated.

3.1.5 Process Calculi

Synchronous Communication

The algebraic representation of process calculi, or process algebra, provides a simple, concise, and powerful way to describe distributed systems. The processes are specified by terms under an algebra. They are constructed from other processes thanks to composition operators (parallel composition, sequence, deterministic or non-deterministic choice, renaming, hiding, ...). The basic processes represent elementary actions, which are most often communication operations (*send* or *receive*). The communication is a directed stream of information: this often relies on the notion of channel which accounts for the different possible streams and a rendez-vous between the *send* operation and a dual operation *receive*. This implies synchronous communication. The following is a transition rule that describes the rendez-vous between two processes P and Q in parallel (symmetric operator $|$) that exchange a message over channel c and then become P' and Q' respectively. $c!$ denotes a send event. $c?$ denotes a receive event:

$$\frac{P \xrightarrow{c!} P' \quad Q \xrightarrow{c?} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

CCS (Calculus of Communicating Systems) [Mil82] is an early and seminal calculus with these features. CSP (Communicating Sequential Processes) [Hoa78] was initially a concurrent programming language that eventually evolved into a process calculus that shares many characteristics of CCS. The communication in these calculi is point-to-point and synchronous. Therefore, it is not adapted to our work. Nevertheless, we make use of CCS in Chapter 5 to specify individual peers. The notion of channel is close to the one we develop in the next chapters which makes it possible to benefit from the composition operators to elaborate complex specifications.

The parallel composition of CCS and synchronous communication are replaced by our model of asynchronous interaction. The parallel composition of CCS and hiding are yet of interest to model internal parallel composition (interleaving) and internal fork-join layouts.

LOTOS (Language of Temporal Ordering Specification) [ISO89] is another example. It is a specification language with a calculus that also proposes similar parallel composition operations of events: per event (“selective parallelism”) or between all observable events (“full synchronisation”). It is an international standard used to describe OSI protocols and systems.

All the previously mentioned calculi are static. This means the algebraic representation of the processes do not evolve according to the content of the messages that are exchanged over channels. Milner has also defined the π -calculus [Mil99]. The main difference is the introduction of parameters: channels can be communicated through channels themselves; new channels can also be created. This allows to describe systems with dynamic configurations. In this work, we do not consider such configurations. Moreover, the π -calculus is also synchronous.

Asynchronous Communication

Nevertheless, there have been many propositions of process calculi that model asynchronous communication. It is possible to represent synchronous communication in asynchronous communication with additional synchronisation messages. It is also possible to describe buffered asynchronous communication with synchronous communication and additional buffer processes.

Some adaptations of the π -calculus have been proposed [HT91, Bou92] to change the semantics of communication into an asynchronous one. A send primitive with no continuation represents asynchronous communication in the asynchronous π -calculus.

Josephs uses a similar approach in [Jos92] and develops an asynchronous process calculus based on CSP with the concept of “receptive processes”. The interactions between a system and the environment are always non-blocking resulting in undefined behaviour when the system is not ready for an input.

In [Pal03], Palamidessi proves that the asynchronous π -calculus is less expressive than the full π -calculus. Beauxis *et al.* [BPV08] compare the asynchronous π -calculus with three different versions of the π -calculus where channels are explicitly represented as special buffer processes. Those buffers are bags, queues or stacks depending on the model of interest. There is a strong correspondence between the asynchronous model and the model with bags. But that correspondence does not hold with queues and stacks. This is conform to our results from Chapter 2: our *Fully Asynchronous* and our *FIFO* models are not equivalent. Since we are interested in comparing the communication models used by the distributed computing community, their π -calculus with queue is not satisfactory. Indeed, the FIFO property is guaranteed for a given channel, whereas distributed algorithms use this property between peers. Since two peers can communicate together via several channels, there is no direct correspondence between these two approaches. We haven’t studied communication models using stacks, since those models are not relevant when dealing with distributed applications, but our approach in Chapter 4 and Chapter 5 allows to specify the duals of FIFO models using stacks instead of queues.

Richer Process Calculi and Mobility

In the π -calculus, the processes are dynamic: the channels can be communicated through channels. If some channels designate processes, this constitutes a first approach of mobility. Mobility means the processes can move between separated domains of computation that may be isolated or interact. The CHemical Abstract Machine [BB92] is a framework that models mobility with notions of molecular reactions in solutions that are isolated by membranes. The reflexive CHemical Abstract Machine enables new kinds of molecules and reactions to take place locally instead

of requiring to travel and react in specific sites which means communication is not centralised in these sites anymore. It fits asynchrony and distribution. The join-calculus [FG96] is a syntactic description of the molecules in the reflexive CHAM. Fournet *et al.* [FG96] also prove it is equivalent to the π -calculus. With the Ambient Calculus [CG98], Cardelli *et al.* focus on the movement of processes called ambients and the boundaries of domains they are allowed to enter or exit. It provides an extensive set of mobility-related primitives to explicitly describe how they enter and exit the domains. The communication between ambients inside a domain is independent of the mobility primitives. In Ambient Calculus, it encodes the asynchronous π -calculus. These calculi are mainly used to model mobility, distribution, firewalls and security properties. However, they do not fit our concerns for two reasons. First, modelling distribution is not straightforward: it usually involves a mix of local communications and moves between domains. Since distribution is at the core of our concerns, we want to keep things as simple as possible. Second, they are not parameterised by communication models. Encoding the communication models would be cumbersome.

Broadcast

Prasad presents CBS, a Calculus of Broadcasting Systems [Pra91] that is based on CCS. Here, the traditional communication rule with rendez-vous between two processes is replaced by rules that describe broadcast communication where one process may broadcast (send) a message on a channel that all the other peers can receive. Even though communication may not always consist of a single transition, this means the communication is still synchronous. The associated communication rules are the following:

$$\frac{P \xrightarrow{c!} P' \quad Q \xrightarrow{c?} Q'}{P|Q \xrightarrow{c!} P'|Q'} \quad \frac{P \xrightarrow{c?} P' \quad Q \xrightarrow{c!} Q'}{P|Q \xrightarrow{c!} P'|Q'} \quad \frac{P \xrightarrow{c?} P' \quad Q \xrightarrow{c?} Q'}{P|Q \xrightarrow{c?} P'|Q'}$$

This implies that every process should be ready to receive messages from the environment (third rule) over every existing channel in order to avoid unspecified behaviours. CBS requires that they are indeed “input-enabled”, even if it consists in stuttering ($P \xrightarrow{c?} P$). This is reminiscent of I/O automata and the property of stability with regard to interest we develop in the next chapter (see Definition 20 on page 63). In this work we consider multicast communication in general whereas CBS describes broadcast communication with peers that are always ready to receive any message. Moreover, CBS is synchronous.

Model Checking for Process Calculi

There exist several model checkers for process calculi. CADP [GLMS13] analyses high-level descriptions written in various languages with synchronous communication, such as LOTOS, LNT (simplified version of E-LOTOS [ISO01]), FSP, or π -calculus. Our goal is not to model check process calculus. Our main concern is the comparison of communication models and being able to check properties over systems with a wide range of communication models.

Overview

Table 3.1 recaps the features of the process calculi that are mentioned previously. None of them perfectly fits the context of our work. However, they may come in handy when it comes to the specification of individual peers (see CCS in Chapter 5).

	Static/Dynamic	Sync/Async	P2P/Multicast
CSP	static	synchronous	point-to-point
CCS	static	synchronous	point-to-point
LOTOS	static	synchronous	point-to-point
π -calculus	dynamic	synchronous	point-to-point
Asynchronous π -calculus	dynamic	asynchronous	point-to-point
Receptive Process Theory	static	asynchronous	point-to-point
Join Calculus	dynamic	asynchronous	point-to-point
Ambient Calculus	mobility	asynchronous	point-to-point
CBS	static	synchronous	broadcast

Table 3.1: Some Process Calculi and their Features

3.2 Asynchronous Communication in Distributed Systems

Ordered delivery has long been studied in distributed algorithms and goes back to Lamport’s paper introducing logical clocks [Lam78]. Implementations of the *FIFO 1-1* and *Causal* communication models using histories or clocks are explained in classic textbooks [Mul93, CDK94, Tel00, KS11, Ray13], and the required information to realise these orders has been studied [PBS89, KS98].

3.2.1 Hierarchy of Ordering Paradigms

Point-to-point asynchronous communication models in distributed systems have been studied and compared. In [CBMT96] and [KS11], they are specified as classes of distributed executions called distributed computation classes in [CBMT96] and message ordering paradigms in [KS11]. They correspond to the exact descriptions of the communication models that are characterised by distributed executions in Chapter 2, that is to say communication models that only rely on the causal partial ordering of communication events (send and receive events). We use the same definition of the causal ordering. Our distributed executions, contrary to the distributed computations, may contain localised internal events which do not change the semantics of the communication models. “non-FIFO” corresponds to the *Fully Asynchronous* communication model; “FIFO” corresponds to the *FIFO 1-1* communication model; “CO (Causally Ordered)” corresponds to the *Causal* communication model.

As in Chapter 2, the hierarchy between these communication models is established from the gradual strengthening of the conditions for two receive events to be causally ordered. Charron-Bost *et al.* [CBMT96] also provide and prove alternative characterisations of the “Causally Ordered computations” (*Causal* distributed executions). They show “CO” computations are equivalent to “MO (Message Ordered)” computations in which two receive events cannot happen in the reverse causal order if the associated send events are causally ordered. They also prove they are equivalent to “EI (Empty Interval)” computations in which no event shall stand between a send event and its associated receive event *when it comes to strict causal ordering* (in the absolute time of a linear extension, events may interleave).

Our work is complementary in two ways. The communication models described in [CBMT96] and [KS11] rely on distributed executions; we consider additional communication models that cannot be characterised by sets of distributed executions. The specifications of these models are based on linear extensions of the causal ordering instead: the runs, as described in Chapter 2. Runs are not distributed computations. The additional communication models are *FIFO n-1*,

FIFO 1-n, and *FIFO n-n*. *FIFO n-n* is often the first step to move away from synchronous communication by decoupling the send and receive events. *FIFO n-1* is sometimes used in the literature [OSB13] without distinction from the classic FIFO order we call here *FIFO 1-1* despite their differences we study in this work.

3.2.2 Hierarchy of Operational Communication Models

In [EMR02], Engels *et al.* establish a hierarchy of communication models for message sequence charts (MSC) that corresponds to ours. The peers are here called instances and are provided with a given type of input and output message buffers. The buffers can be global, local to an instance, associated to a pair of instances, or local to each message. Depending on the configuration of the buffers for input and output, a great variety of communication models arises: it is reduced by equivalence to a smaller set that includes the ones we study. The intuitions and examples of implementations using buffers we present in Chapter 2 are reminiscent of the work in [EMR02]. Table 2.1 in Chapter 2 associates a communication model to each couple of ordering type (among causal, local, and global) on send and receive events. This corresponds to the layout of buffers in [EMR02] and the event orderings they rely on. Although similar, our work focuses on logical specifications (Chapter 2) of the communication model and operational specifications based on message histories (Chapter 4) while Engels *et al.* [EMR02] compare operational and practical implementations using buffers. In particular, the *Causal* communication model do not emerge from any layout of buffers, thus, it is not part of the hierarchy in [EMR02]. In our work, we provide homogeneous logical (Chapter 2) and operational (Chapter 4) descriptions of the communication models that make it possible to compare all the models and establish a complete hierarchy. In [EMR02], Engels *et al.* consider an additional communication model called “inst2” that is purely operational and requires an intermediate step between send and receive to transfer a message through the appropriate buffers. It corresponds to a combination of *FIFO 1-n* and *FIFO n-1* we can specify thanks to the concept of composite communication models that we develop in Chapter 5.

3.2.3 Realisability with Synchronous Communication

Charron-Bost *et al.* [CBMT96] and Kshemkalyani *et al.* [KS11] introduce the concept of realisability with synchronous communication. It is not exactly the *RSC* communication model presented in Chapter 2 that we study in this work. In [CBMT96] and [KS11], it characterises distributed executions whereas we characterise runs: a distributed execution is realisable with synchronous communication when there exists a non-separated linear extension (run) of the underlying causal ordering. In a “non-separated run”, as in “EI (Empty Interval)” computations, no additional event shall interleave between a send event and the associated reception. Ignoring internal actions, this corresponds to our actual specification of the point-to-point *RSC* model. As for *Causal*, Charron-Bost *et al.* provide alternative characterisations of *RSC*. In particular, they highlight the relations with actual synchronous communication. The *RSC* computations are shown to be equivalent to computations that may be represented by space-time diagrams with only vertical lines and computations that do not contain crowns. Given n couples of associated send and receive events s_1, \dots, s_n and r_1, \dots, r_n : they form a crown if $\forall i \in 2..n : s_{i-1}$ precedes r_i but s_n precedes r_1 .

3.2.4 Summary

Table 3.2 sums up the equivalence between our communication models and the literature: it reveals that we unify and complete existing hierarchies.

Model in [CBMT96] and [KS11]	Model in [EMR02]	Model in our work
Non-separated linear extensions of computations	<i>nobuf</i>	<i>RSC</i>
	<i>global</i>	<i>FIFO n-n</i>
	<i>inst2</i>	<i>FIFO 1-n and FIFO n-1</i>
	<i>inst_{out}</i>	<i>FIFO 1-n</i>
	<i>inst_{in}</i>	<i>FIFO n-1</i>
CO (Causally Ordered)		<i>Causal</i>
FIFO	<i>pair</i>	<i>FIFO 1-1</i>
non-FIFO	<i>msg</i>	<i>Fully Asynchronous</i>

Table 3.2: Communication Models of the Hierarchies found in the Literature and This Work. A blank cell means the model is not specified and not part of the hierarchy.

Furthermore, our work takes multicast communication into account and adapts the definition of *RSC* accordingly. We also prove the multicast counterpart of the hierarchy of communication models and highlights the differences with the extended point-to-point hierarchy.

3.3 Formal Verification

There are different approaches to the formal verification of distributed systems that satisfy three main needs:

- Verifying the validity of existing distributed systems and algorithms. This means proving that existing systems satisfy their specifications. The verification is often static and relies on proof assistants with a varying degree of *ad-hoc* automation.
- Designing distributed systems that are correct by construction. In other words, deriving systems from their specifications. This usually relies on a top down approach with step-wise mechanised refinement, possibly supplemented by certified translation to source code. Refinement is at the heart of the design and comparison of the diverse specifications of communication models in Chapter 6.
- Detecting and avoiding bugs in running implementations of distributed systems. In practice, implementations of distributed system might not have been constructed or proven to be bug free and errors might occur after long running sessions. Runtime verification aims at detecting or avoiding such errors at runtime. For example, during execution of the system, the CrystalBall [YKKK09] approach consists in locally model checking a system for safety violation over a state space that depends on recent snapshots of the neighbouring peers. The state exploration predicts and avoid redundant interleaving of causally independent chains. This sneak-peek at the future enables a peer to “steer” away from a path that would eventually lead to safety violation. We do not focus on runtime verification in our work and will not detail this approach any further.

3.3.1 Proof Assistance

Manual Proof

Formal verification of distributed algorithms has been conducted with success. However the hypotheses on the communication are often fuzzy or unclear and one has to dive deep into the proofs to identify them. For instance, [LMP04] studies the topology maintenance in structured peer-to-peer networks. Different algorithms are studied, some assume FIFO channels and some do not. It is unclear why it is required, and if it is required for all channels.

[LMW11, Lu13, AMW17] presents a full and complete verification of Pastry, an overlay and routing protocol for the implementation of a distributed hash table, in TLA^+ , which has been mechanically checked with the TLA^+ Proof System [CDLM10]. It assumes asynchronous communication with loss of message, and uses send and receive actions similar to the one presented in this paper. Part of the complexity of the proof is the interleaving of actions caused by the *Fully Asynchronous* delivery of messages, and it would be interesting to find if a more orderly communication (e.g. *FIFO 1-1* or *Causal*) would simplify it. Moreover, when the formal proof only considers safety properties, the loss of messages has no influence. [Zav12] models Chord, another algorithm for a distributed hash table, in Alloy but it contains no explicit communication actions. Actually, distributed communication is modelled by means of shared state: peers are allowed to read the states of other peers. The paper argues that this avoids many implementation details while preserving the central concepts of Chord.

Assistance to the Writing of Proofs

Given the implementation of a distributed system, it is easier to prove it correct under a simple reliable environment than under a more realistic fault model. Wicox *et al.* propose Verdi [WWP⁺15], a framework that makes it possible to transform an implementation and proofs of safety in Coq established under an ideal environment into an implementation that is fault-tolerant under a more hostile environment. “Verified System Transformers” may enrich the system in a modular way with fault models for crashes, message loss, duplication or reordering.

Implementing and proving systems in Coq in the first place still demands efforts. Ricketts *et al.* [RRJ⁺14] describe the domain-specific language REFLEX and adapted Coq proof tactics that have been designed alongside and alleviate the burden of manual proof in reactive systems.

3.3.2 Correct-by-Construction Design of Distributed Systems

Refinement

In this work, in Chapter 6, we mechanise an extensive set of proofs of refinement between specifications of communication models with the Event-B method. This work has also partially been conducted in TLA^+ with the TLA^+ Proof System. Both methods offer their own specific take on the concept of refinement. Chapter 5 and Chapter 6 respectively provide in-depth presentations of the TLA^+ specification language and the Event-B method.

The two classic characteristics of refinement of an abstract system by a concrete system are the weakening of the preconditions and the compliance to the abstract system. Informally, this means the concrete system might extend the abstract system with additional behaviour or become more deterministic.

TLA⁺

In TLA⁺, the specification of a system (initial state and transition predicates) may also include fairness properties. The refinement of a TLA⁺ specification by another is the logical implication $Spec_c \Rightarrow Spec_a$ where $Spec_c$ and $Spec_a$ are the concrete and abstract specifications, including fairness. Notably, this means the concrete specification shall not introduce more deadlocks. Sometimes, the structure of the transition predicates, called “actions” in TLA⁺, is similar enough in both specification to allow for a refinement of the actions individually. In our work, all the specifications for the communication models consist of two actions “send” and “receive”. A model refines another when the predicate of the concrete “send” action implies the predicate of the abstract “send” action and the concrete “receive” implies the abstract one.

In [Lam11, Lam10], Lamport describes the addition of Byzantine resilience to standard Paxos. The proof is conducted by refinement of the distributed non-Byzantine algorithm and has been mechanically checked with the TLA⁺ Proof System [CDLM10]. Another approach is presented in [Lam01]. Three versions of Paxos (the classic one, disk Paxos and Byzantine Paxos) are derived from an abstract, non-distributed algorithm.

Event-B

Unlike refinement in TLA⁺, refinement of Event-B specifications called “machines” always consists of a refinement of the operations called “events”. The guards may be weakened and the behaviour must conform to the abstract event. New events refine the special event called “skip”. Variants might be used to avoid divergence. There is no fairness in Event-B.

The Event-B book [Abr10] presents several examples of refinements of distributed algorithms. The “simple file transfer protocol” decomposes the atomic sending of a file in a sequence of send events, and uses counters to coordinate the progression. This protocol is later extended to handle loss and re-transmission with an alternating bit protocol. In this example, asynchronous communication appears implicitly during refinement, and properties of the communication are directly embedded in the resulting machine. A logical clock is used in the “routing algorithm for a mobile agent”. It is used to order the messages sent by a mobile agent while it moves. This example can be seen as the development of an ordered communication model down to a concrete model that can be localised. Lastly, the “leader election on a connected graph network” (also in [ACM03]) deals with the difficulties of splitting an atomic action (in a shared-memory model) into several actions (in a message-passing model). This creates deadlocked states (a situation called “contention” in the algorithm) where two nodes are each waiting for the other to progress. This development is more concerned with providing a algorithmic solution in presence of non-atomic actions, than with the development of non-atomicity (i.e. messages).

[AMS12, AMS14] presents the development of snapshot algorithms with Event-B. A context NETWORK describes the static part (processes and channels). A machine SYSTEM specifies the send, receive and internal events. The development is done by refinement, starting with the specification of the snapshot problem, which is by essence a global property. A generic architecture with asynchronous communication is presented, which allows the derivation of several algorithms. At one point, the set of messages (which models fully asynchronous communication) is refined by FIFO queues (which models ordered communication). This leads to another, simpler, snapshot algorithm, which ends being the well-known Chandy-Lamport algorithm.

[RRM05] presents the development in Event-B of a topology maintenance algorithm for fault-prone network. The abstract specification is described with two global variables, refinement introduces events and splits the variables on each peer. At no point messages are explicitly introduced and it is thus unclear what is the exact communication model that is required. [SY10] presents the development of a total order broadcast using Event-B. Communication is

explicitly modelled by a broadcast event. The presented algorithm uses a unique sequencer which numbers messages. As the goal is to develop a Byzantine immune algorithm, no hypothesis is made on the communication. [ILTR11] describes the formal derivation of an algorithm for leader election in Event-B. The abstract model is centralised, and refinement introduces distribution. The behavioural part of the communication model first comprises two events, *send* and *receive* which directly access the state variable of the other peers. Then, a new refinement introduces new variables to decouple the peers and to get a “one-to-one asynchronous communication channel”.

[Bry11] presents the development of a consensus algorithm using stepwise refinement. The last refinement introduces messages and send and receive events. The studied algorithm relies on a synchronous timing model and is round-based, meaning that all messages sent in one round are received at the end of this same round. Thus, the model splits a round in three successive phases: “sending” (processes send messages), “receiving” (getting and handling all the messages for each process), and “restarting” (resetting the state of the processes for the next round). The transition between the phases are still global specifications, even if the paper suggests that this could be further refined in local events.

[BTMK16] shows how to combine refinement and composition to transform a distributed algorithm with local termination detection into an algorithm with global termination detection. Their approach uses refinement to traditionally develop the algorithm, and then to introduce the last step for global termination detection. This global termination detection is achieved with a proved model of a global termination detector. Certified composition is used to combine the detector and the algorithm. The correct-by-construction approach ensures that the extended algorithm satisfies the same specification as the original one.

Translation and Code Generation

In this work, we do not consider code generation. The work carried in Chapter 6 where we refine each communication model through advanced degrees of concretisation does not encompass code generation because our goal is actually to compare the different specifications of the communication models. Nevertheless, there are many approaches to code generation in formal verification.

For example, Tounsi *et al.* present B2Visidia [TMM16], a tool that generates runnable Java programs from the formal specification of a distributed algorithm in a subset of the Event-B language. Thanks to annotations of models specified with the Event-B language that make it compatible with the B2Visidia language, an adapted subset of the Event-B language, the tool generates Java code that can be executed by the Visidia [BM03] software (a framework for implementation and experimentation on distributed algorithms).

Mace [KAB⁺07] is a specification language and C++ implementations generator. Mace is a C++ language extension that restricts how systems can be specified and allows to specify the different layers. This results in high-performance implementations that preserve high level structures which makes it possible to benefit from features such as model checking of Mace code. For example, MaceMC [KAJV07], a model checker whose key feature deals with finding safety properties that might hide behind liveness properties, is used on Mace specifications.

Splitting the specification of a distributed system or protocol into different layers is also at the heart of the IronFleet [HHK⁺15] methodology. On the “distributed protocol” layer, verification consists in classic refinement from the higher level specification. On the “implementation” layer, the verification is Hoare-like and performed using Dafny [Lei10], a program verifier that can generate C# programs.

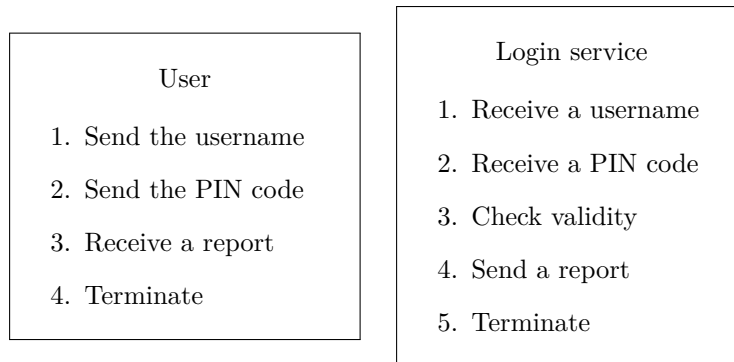
Part II

Point-to-Point Communication

Chapter 4

Compatibility Checking of Communicating Peers

The objective is to check the compatibility of the composition of a set of peers, given a behavioral description of the peers and a communication model. To get an intuition, consider the compatibility in terms of termination of the composition of two peers specified as follows:



In the synchronous world, the compatibility of these two peers is well defined: both match on a first rendez-vous on the username, then proceed to a second rendez-vous on the PIN code, then on the report, and eventually terminate. However, this is less clear in the asynchronous world. Traditionally, from a distributed systems point of view, one considers that the communication medium controls the message deliveries: this means it pushes messages up to the applications. The applications cannot impose a delivery order. In our example, if the communication medium ensures *FIFO* ordering, then the username is delivered before the PIN code (because they are sent in this order) and both peer terminate: we say they are compatible. However, if the communication medium is *Fully Asynchronous*, the PIN code may be delivered first. The login service might be unable to cope with such a situation. From there, termination of the login service is uncertain. Moreover, there is no guarantee that a report would ever be sent: the user could wait for it and also never terminate. Whether a positive output exists or not, the composition is deemed incompatible under *Fully Asynchronous* communication.

Checking the compatibility could consist in extensively building the set of all the runs of the composition of peers, reducing it by filtering out the runs that are irrelevant in the chosen communication model (according to the definition from Chapter 2), and eventually evaluating

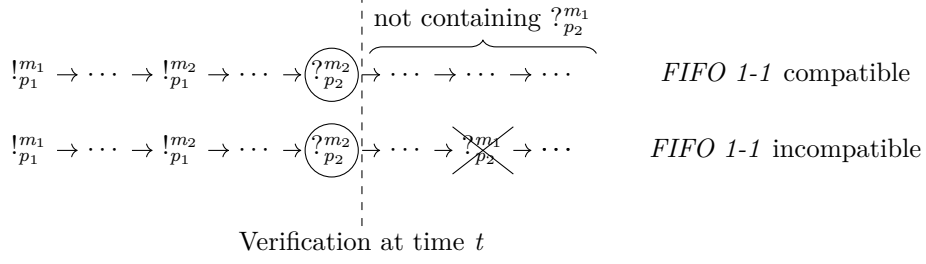


Figure 4.1: Two Runs that Illustrate the Oracle Issue in Compatibility Checking. The validity of the reception of m_2 on p_2 depends on the future in the run.

properties on this set (non-emptiness, temporal properties, ...). Although this solution would make use of the exact definitions of the communication models in Chapter 2, this is not satisfactory because they are about whole runs. They are actually oracles. A reception is indeed valid at a given point with regard to all events: events that happen before and *after* that reception. For instance, consider *FIFO 1-1* and two messages m_1 and m_2 sent by the same peer in this order: the reception of m_2 on a peer p at time t is correct only if p never receives m_1 later at time $t' > t$. This means that the verification of compatibility at time t depends on events that have yet to happen. Figure 4.1 illustrates this example. The verification of compatibility has to be of practical use: a communication model must decide to deliver a message on past events only. Therefore, our description of the communication models should be operational instead. It should also remain as abstract as possible, not to preclude implementations, so that the compatibility can be ascertained for any practical implementation. The chosen formalisation is based on message histories: they contain information about the formerly sent messages that are of interest to build valid runs.

The specifications of the peers of a composition, the communication models, the overall system in which they interact, and the compatibility properties constitute a framework for the verification of the compatibility of communicating peers. The first section describes and formalises this whole framework. The second section demonstrates its validity in terms of correctness and completeness.

Correctness The runs generated by the framework match the definitions of the communication models in Chapter 2.

Completeness The framework generates all possible runs that match these definitions and the behaviour of the peers that are involved in the composition.

A peer is specified by a transition system labeled with internal or communication events. The correctness and completeness of the framework rely on well behaved peers that prevent the communication model from having to glimpse into the future. Since a communication model is also specified by a transition system labeled with communication events, modelling the communication simply consists in an operation close to a synchronous product of labeled transition systems: several peers and a communication model. The runs are the sequences of transitions of the traces of the resulting system. Checking the compatibility of a composition of peers under a communication model consists in evaluating the properties of a given compatibility criterion in this system.

The mechanisation of this framework in TLA^+ is described in the next chapter.

4.1 Description and Formalisation of the Framework

4.1.1 Channels

Let \mathcal{C} be an enumerable set of channels. A channel is a label on messages. The content of messages, that is to say the data they carry, is irrelevant outside the scope of peers' internal behaviour. Going back to the introduction example with the user and the login service, the actual username, the actual PIN code, and the content of the report do not matter at all when it comes to the communication and compatibility checking. This information is not accessible to the communication model, as if inside an opaque parcel. The channel of a message plays the role of a label on the opaque parcel: it enables the communication model to distinguish messages that would otherwise look identical from this external viewpoint. In the framework, messages do not have an explicit destination peer. The peers send messages through channels instead. In our example, the user would send its username on a channel `username`, its PIN code on a channel `PIN`, and the login service would report on a `report` channel. Here, the different channels make it possible to decide on the compatibility: under the *Fully Asynchronous* communication model, it reveals that the delivery of the username and PIN code in the reverse order poses a problem because the specification of the login service refers to a specific reception order from the two channels.

The channels are not restricted to one sender and one receiver. Different peers can send messages on the same channel. Likewise, different peers can receive a message from the same channel. Yet, it is nonetheless a point-to-point communication abstraction because a given message still has exactly one sender and at most one receiver. If a single message is sent on a given channel and several peers expect to perform a reception from this channel, only one of them will be able to receive the message. This allows for richer and more elegant system specifications where a message can be received by a peer that depends on the state of the communication medium. For example, it is possible to describe arbitrary client-server and publish-subscribe architectures naturally.

4.1.2 Specification of Compositions of Peers

Specification of Individual Peers

Peers are specified using transition systems labelled by communication events or internal actions. This provides simplicity and flexibility in theoretical and practical uses. It is adapted to verification techniques like model checking. Optionally, in order to ease the specification of peers, the transitions systems can be derived from process calculi terms such as CCS [Mil82] as detailed in Section 5.3.1 in the next chapter. Communication models are also specified using transition systems. The communication interactions are detailed in 4.1.4.

Definition 18 (Peer Specification). *A peer is specified by a labeled transition system (S, I, R, L) where:*

- S is the set of states.
- I is the set of initial states: $I \subseteq S$.
- L is a (enumerable) set of labels: $L \subseteq (\{\text{Send}, \text{Receive}\} \times \mathcal{C}) \cup \text{Internal}$.
- R is the transition relation: $R \subseteq S \times L \times S$.

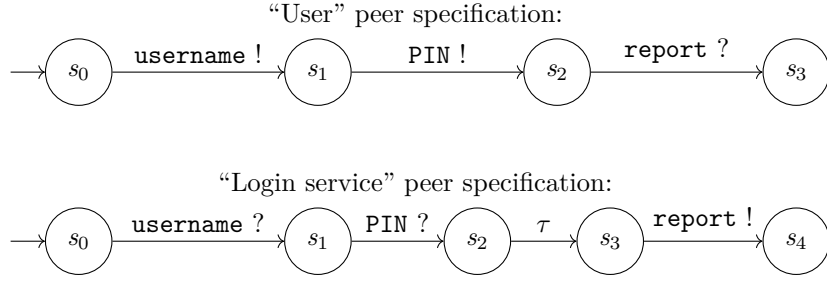


Figure 4.2: Representation of the Specification of the two Peers from the Introduction Example

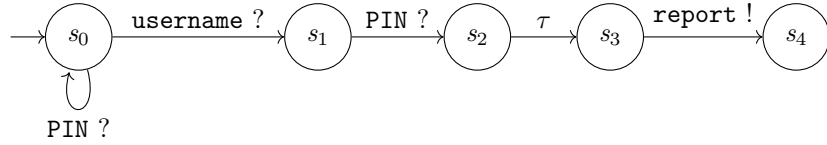


Figure 4.3: A Specification of the Login Service that discards messages from the PIN channel until a username is received.

Given a channel c , the labels (Send, c) and (Receive, c), alternatively denoted $c!$ and $c?$, are interpreted as the sending of a message on channel c , and the reception of a message from channel c . Internal is interpreted as any other internal action that does not involve communication and alternatively denoted τ .

Figure 4.2 provides a graphical representation of the transition systems that specify the two peers of the introduction example: the user and the login service.

Listened Channels

As mentioned in the introduction of this chapter, the communication medium controls the message deliveries. This is why the composition of the two peers of the running example is incompatible under certain communication models that might push unexpected messages. In order to solve this issue, the specifications of the peers could be strengthened to handle the edge cases. Consider the specification of the login service depicted in Figure 4.2. Under *Fully Asynchronous* communication, a message from the PIN channel might be pushed towards the login service when it expects a message from the `username` channel in state s_0 . A very simple workaround consists in discarding all the messages from the PIN channel until an actual username is received. The specification is an additional $(s_0, \text{PIN ?}, s_0)$ transition as shown in Figure 4.3. Being ready to handle all kinds of messages makes sense in this situation. If the composition involved other peers that exchanged unrelated data through independent channels, there would also be a risk to deliver unexpected messages to the user and login service. However, this time, making sure the login service is able to cope with all the messages that may transit through the communication medium is inappropriate: it would mean the specification of a peer should change according to the exact environment it takes part in. The communication medium should not push such messages that belong to a different sub-environment of communication. This problem arises from the absence of an explicit destination peer on each message.

The concept of listened channels provides a sensible compromise between adapting the specifications to the environment and explicit destination peers. The communication medium pushes

a message up to a peer only if a reception from the channel on which this message transits is specified in the current state of the peer. This means that in state s_0 of the peer from Figure 4.3, the communication medium can push messages in transit on channels **username** and **PIN** but it may behave as if messages in transit on other channels did not exist.

Definition 19 (Listened Channels). *Let $P \triangleq (S, I, R, L)$ be the specification of a peer and s a state in S . The listened channels $LC_P(s)$ of P in s is the set of channels involved in possible receptions from s .*

$$LC_P(s) \triangleq \{c \in \mathcal{C} \mid \exists s' \in S : (s, (\text{Receive}, c), s') \in R\}$$

There is however a caveat. Going back to the specification P_{login} of the login service in Figure 4.2, **username** is the only listened channel in s_0 : $LC_{P_{\text{login}}}(s_0) = \{\text{username}\}$. Considering that the communication medium, depending on the properties it models, may behave as if messages transiting on non-listened channels did not exist, this means that the login service chooses a delivery order. This is unrealistic and negates the purpose of compatibility checking.

The specification of the login service in Figure 4.2 is legal yet not suited for compatibility checking unlike the alternative specification P'_{login} in Figure 4.3. In the latter, $LC_{P'_{\text{login}}}(s_0) = \{\text{username}, \text{PIN}\}$. Channel **PIN** is indeed expected to be listened to since the login service has future interests in it. The absence of **username** in $LC_{P'_{\text{login}}}(s_1) = \{\text{PIN}\}$ does not pose a problem because from state s_1 , the login service is not interested in messages transiting on **username** anymore. This channel is then no different from any other independent channel the peer does not listen to. We say that such specifications are stable with regard to interest: the set of listened channels (in states involving receptions) can only decrease. If a peer is not interested in a channel in a given state (i.e. it is not listening to it), then it will never be interested in it later.

Definition 20 (Stability with regard to Interest). *The specification $P \triangleq (S, I, R, L)$ of a peer is stable with regard to interest if and only if the set of listened channels decreases on states involving receptions.*

$$\forall s, s' \in S : (s, s') \in R^+ \Rightarrow \left(\begin{array}{ll} LC_P(s) = \emptyset & \text{(no receptions in } s) \\ \vee \quad LC_P(s') \subseteq LC_P(s) & \text{(less channels of interest)} \end{array} \right)$$

R^+ is the transitive closure of $\{(s, s') \in S^2 \mid \exists l \in L : (s, l, s') \in R\}$.

4.1.3 Specification of Communication Models

The specification of a communication model is a labelled transition system that is similar to the specifications of peers. The transitions are internal or communication events on channels. There is a difference from the specifications of peers: events are localised on peers. A reception is also characterised by a set of listened channels. This may be taken into account depending on the properties of the communication.

Definition 21 (Communication Model Specification). *A communication model is specified by a labeled transition system (S, I, R, L) where:*

- S is the set of states.
- I is the set of initial states: $I \subseteq S$.

- L is an enumerable set of labels:

$$L \subseteq \left(\begin{array}{l} \text{Internal} \\ \cup \quad \mathcal{P} \times \{\text{Send}\} \times \mathcal{C} \\ \cup \quad \mathcal{P} \times \{\text{Receive}\} \times \mathcal{C} \times 2^{\mathcal{C}} \end{array} \right)$$

- R is the transition relation: $R \subseteq S \times L \times S$.

Given a peer p , a channel c , and a set of channels C , the labels (p, Send, c) and $(p, \text{Receive}, c, C)$, alternatively denoted $p, c!$ and $p, c?, C$, are interpreted as p sending a message on channel c , and p receiving a message from channel c while listening to channels in C . Internal is interpreted as any other internal action of the communication model that does not actually involve any interaction with the peers. It is alternatively denoted τ .

Figure 4.4 depicts a simple communication model that handles, without any particular ordering, messages on channels `username` and `PIN`. It is limited to two messages in transit. s_0 corresponds to the state without any message in transit, a message on each channel in s_5 , two messages on `PIN` in s_4 , and so on. The actual definition of a communication model depends on its characteristics and examples are provided in Section 4.1.6.

4.1.4 Overall Product System

Definition 22 (Specification of a System). Let $(P_p)_{p \in \mathcal{P}} \triangleq (S_p, I_p, R_p, L_p)_{p \in \mathcal{P}}$ the specifications of the peers in \mathcal{P} and $\text{CM} \triangleq (S_{\text{CM}}, I_{\text{CM}}, R_{\text{CM}}, L_{\text{CM}})$ the specification of the communication model.

The specification of the overall system consisting of the composition of peers under this communication model is the product of the $(P_p)_{p \in \mathcal{P}}$ and CM with synchronisation on the send (resp. receive) transitions of a peer and the communication model. It is a transition system (S, I, R) where:

- S is the set of states:

$$S \triangleq S_{\text{CM}} \times \prod_{p \in \mathcal{P}} S_p$$

- I is the set of initial states:

$$I \triangleq I_{\text{CM}} \times \prod_{p \in \mathcal{P}} I_p$$

- R is the transition relation:

$$R \triangleq \left\{ \begin{array}{l} (s, s') \in S \times S \\ \text{where:} \\ s \triangleq (s_{\text{CM}}, (s_p)_{p \in \mathcal{P}}) \\ s' \triangleq (s'_{\text{CM}}, (s'_p)_{p \in \mathcal{P}}) \end{array} \right\} \vee \left\{ \begin{array}{l} \left(\begin{array}{l} \text{Stuttering} \\ s = s' \end{array} \right) \\ \vee \left(\begin{array}{l} \text{Internal action} \\ (s_{\text{CM}}, \text{Internal}, s'_{\text{CM}}) \in R_{\text{CM}} \\ \wedge \quad \forall p \in \mathcal{P} : s_p = s'_p \end{array} \right) \\ \vee \left(\begin{array}{l} s_{\text{CM}} = s'_{\text{CM}} \\ \wedge \quad \exists p \in \mathcal{P} : \\ (s_p, \text{Internal}, s'_p) \in R_p \\ \wedge \quad \forall q \in \mathcal{P} \setminus \{p\} : s_q = s'_q \end{array} \right) \\ \vee \left(\begin{array}{l} \text{Communication} \\ \exists p \in \mathcal{P} : \exists c \in \mathcal{C} : \\ \text{send}(s, s', p, c) \\ \vee \text{receive}(s, s', p, c) \end{array} \right) \end{array} \right\}$$

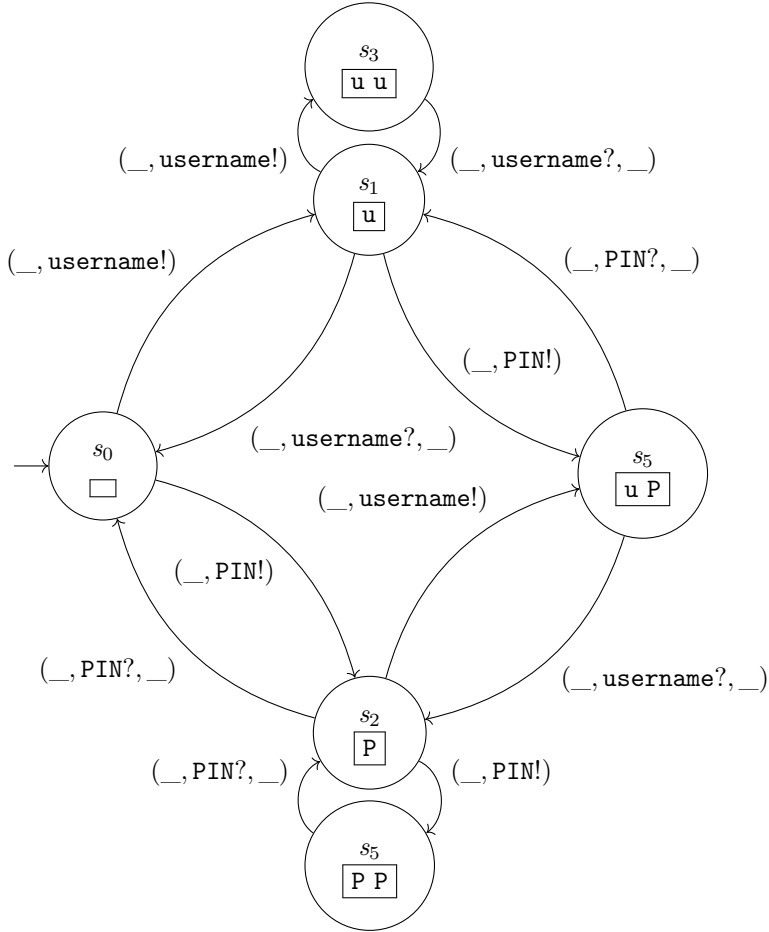


Figure 4.4: Specification of a *Fully Asynchronous* Communication Model with Two Channels and Limited to Two Messages in Transit. The two channels are **username** and **PIN**. Here, **_** in a tuple means any possible value for the peer or set of listened channels. In each state, messages in transit are depicted in a box: **u** for messages transiting on **username**, **P** for messages transiting on **PIN**.

where the rendez-vous on send actions in peer p and the communication model is:

$$\text{send}(s, s', p, c) \triangleq \left(\begin{array}{l} \left(s_p, (\text{Send}, c), s'_p \right) \in R_p \\ \wedge \left(s_{\text{CM}}, \left(p, (\text{Send}, c) \right), s'_{\text{CM}} \right) \in R_{\text{CM}} \\ \wedge \forall k \in \mathcal{P} \setminus \{p\} : (s_k, \text{Internal}, s'_k) \in R_k \end{array} \right)$$

and the rendez-vous on receive actions in peer p and the communication model is:

$$\text{receive}(s, s', p, c) \triangleq \left(\begin{array}{l} \left(s_p, (\text{Receive}, c), s'_p \right) \in R_p \\ \wedge \left(s_{\text{CM}}, \left(p, (\text{Receive}, c), \text{LC}_{P_p}(s_p) \right), s'_{\text{CM}} \right) \in R_{\text{CM}} \\ \wedge \forall k \in \mathcal{P} \setminus \{p\} : (s_k, \text{Internal}, s'_k) \in R_k \end{array} \right)$$

Parallel independent communication cannot happen: sending and receiving a message is a rendez-vous between a send or receive transition of *one* peer and the communication model. Contrary to synchronous communication between peers where a rendez-vous occurs between a send transition and a receive transition, the send and receive operations are here *distinct*.

In Figure 4.5, we consider simplified specifications of the user and login service: we focus on the transmission of the username and PIN code only. The login service is stable with regard to interest. It is ready to receive messages in any order from both channels. The figure shows the specification of the composition of the two peers under the communication model from Figure 4.4. This communication model does not guarantee that messages are received in any particular order. Hence, there are two branches in the system: one in which the username is received before the PIN code, one in which the PIN code has been received before and discarded by the login service. The verification of the compatibility of the composition under the particular communication model consists in verifying if properties of interest are satisfied by this system.

4.1.5 Compatibility Checking

Compatibility depends on specific criteria. The same composition under the same communication model might be compatible or not depending on the selected criterion. In the running example depicted in Figure 4.5, several compatibility criteria (and combination of criteria) are significant:

1. Does the user eventually terminate?
2. Does the login service eventually terminate?
3. Do all the peers eventually terminate?
4. Are there messages eventually left in transit?

Termination of the user means it has reached state u_2 of its specification, after all messages have been sent. Termination of the login service means it has reached state l_2 of its specification, after all messages have been received. This is an explicit specification decision. Assuming minimal progress, the overall system eventually ends up either in state (s_0, u_2, l_1) or (s_0, u_2, l_2) . In both cases, it corresponds to the user in u_2 (termination) and the communication model in s_0 (the state corresponding to an empty network of messages in transit). However, the login service can be stuck in state l_1 : this happens when the message on PIN is received first.



Overall system under the communication model from Figure 4.4:

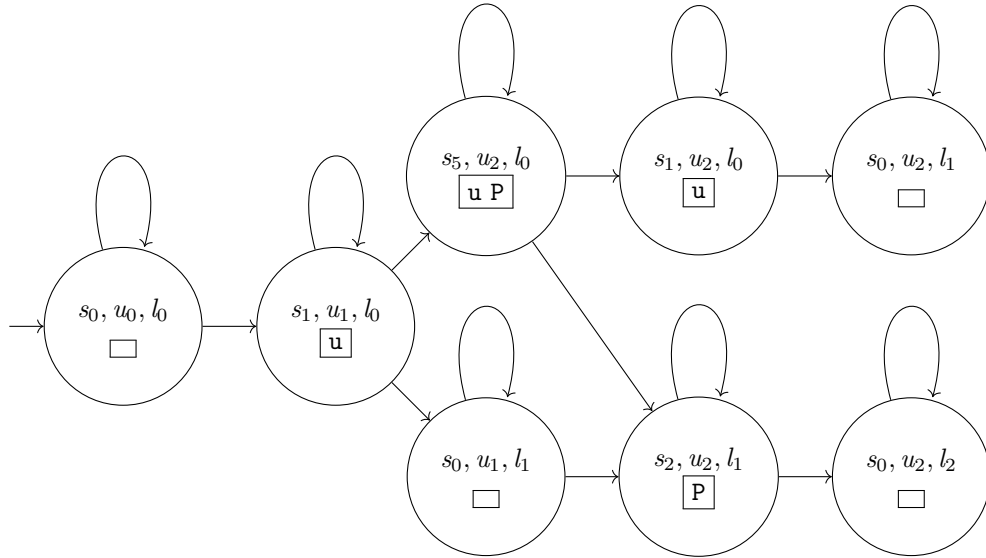


Figure 4.5: Transition system associated to the composition of simplified versions of the user and login service under the communication model from Figure 4.4. According to the definition of an overall system, a state is a tuple whose first element corresponds to a state of the communication model, the second and third to states in the two specifications of the peers. In each state, messages in transit are depicted in a box: **u** for messages transiting on **username**, **P** for messages transiting on **PIN**.

1. The user eventually terminates: **OK**.
2. The login service **may not** terminate.
3. A peer **may not** terminate.
4. Eventually, there are not any messages left: **OK**.

Had the peers been composed under a communication model that imposes that messages are delivered in the order of their emission (such as *FIFO 1-1*), the branch containing states (s_5, u_2, l_0) , (s_1, u_2, l_0) , and (s_0, u_2, l_1) would not exist in the system and all four compatibility properties would be satisfied.

Specifying termination requires to identify subsets of states in the specifications of peers. Similarly, identifying faulty states following unexpected receptions makes sense. For example, when the PIN code is received before the username, the login service remains in the same state hoping to function normally from then. It could instead switch to a separate faulty state and stop doing anything else. Once a peer has reached a terminal or faulty state, it shall remain in it and stop sending and receiving messages. Consequently, distinguishing between different terminal (resp. faulty) states is not necessary and we therefore collapse them into one terminal state and one faulty state usually denoted 0 and \perp .

Additional compatibility properties include the absence of communication deadlocks. Strictly speaking, termination or the faulty state are cases of deadlock that are handled by dedicated compatibility criteria. A communication deadlock consists in reaching a stable state that is not termination or fault. In the running example, state (s_0, u_2, l_1) corresponds to a communication deadlock: a message on PIN is expected by the login service but the communication model never provides that message. This compatibility property makes sense in systems that are not supposed to stop.

Definition 23 (Compatibility Properties). *A compatibility property is given as an LTL formula over a system. Let $\text{System} \triangleq (S, I, R)$ be a system and $s \triangleq (s_{\text{CM}}, (s_p)_{p \in \mathcal{P}})$ a state in S . Let 0 and \perp denote the terminal and faulty states in the specifications of the peers.*

The following state-wise properties are defined:

- *All the peers are in the terminal state:*

$$0_{\forall} \triangleq \forall p \in \mathcal{P} : s_p = 0$$

- *Peer p is in the terminal state:*

$$0_p \triangleq s_p = 0$$

- *An unexpected message has been delivered:*

$$\perp_{\exists} \triangleq \exists p \in \mathcal{P} : s_p = \perp$$

The compatibility properties include the following classic temporal properties the system may satisfy assuming minimal progress:

System termination *The system always reaches the terminal state:*

$$\text{System} \models \Diamond \Box 0_{\forall}$$

Peer termination *Peer p always reaches the terminal state:*

$$System \models \Diamond \Box 0_p$$

No faulty receptions *No unexpected reception ever occurs:*

$$System \models \Box \neg \perp_{\exists}$$

Absence of communication deadlock *No state is stable except termination or fault:*

$$System \models \left(\begin{array}{l} \forall s \in S : \Box \Diamond \neg s \\ \vee \quad \Diamond \Box 0_{\forall} \\ \vee \quad \Diamond \Box \perp_{\exists} \end{array} \right)$$

The list of compatibility properties is not exhaustive. Ad-hoc state or temporal properties might be of interest.

4.1.6 Specification of Communication Models with Message Histories

Specifications for communication models (Definition 21) corresponding to the seven communication models from Chapter 2 are provided in this section. The conformance to the reference definitions of the communication models in Chapter 2 is studied in the next section.

Common Layout

All the specifications follow a uniform layout that relies on a data structure called “message history”. A message carries a message history: it contains previous messages that might help enforce the ordering properties of a communication model. The state of a communication model consists of:

- a network (a set) of messages in transit,
- either a global history (common to all the peers) or localised histories (one per peer) of messages serving as a reference for the histories of newly introduced messages.

More precisely, a message is comprised of:

- the channel on which it has been sent,
- its sender,
- the set of previously sent messages it depends on (the history).

The reception of a message is possible if its history does not contain a message that is still in transit, with mitigations on the exact conditions depending on the ordering policy. For example, the restriction may be applicable to messages that have been sent by the same peer only (*FIFO 1-n*). Especially, the listened channels filter out forbidden receptions in several communication models. In other communication models, they have no influence.

Building Message Histories

Initially the global (resp. local histories) are empty. A message carries the content of the global history (resp. local history of the sender) at the time of emission. Once the message is sent, the global history (resp. local history of the sender) carries that message. For example, consider a communication model with a global history H containing two messages m_1 and m_2 and a network net containing m_2 . The following diagram shows the evolution of the state variables when a new message is sent by peer p on channel c .

$$\begin{array}{ccc} net = \{m_2\} & \xrightarrow{p, c!} & net = \{m_2, (c, p, \{m_1, m_2\})\} \\ H = \{m_1, m_2\} & & H = \{m_1, m_2, (c, p, \{m_1, m_2\})\} \end{array}$$

When a message is received, the network evolves (the message in question is removed) but the histories remain unchanged. In the specification of the *Causal* communication model (relying on local histories) however, more messages are added to the history of the receiver during the reception. This models the aspect of causal dependency that corresponds to the transmission of a message from one peer to another. The message that is received *and* the messages carried in its history are added to the history of the receiver. For example, consider the following state in the *Causal* communication model and the reception of a message from channel c on p_3

$$\begin{array}{ccc} m_1 \triangleq (a, p_1, \emptyset) & net = \{m_3\} & net = \emptyset \\ m_2 \triangleq (b, p_1, \{m_1\}) & H_1 = \{m_1, m_2\} & H_1 = \{m_1, m_2\} \\ m_3 \triangleq (c, p_2, \{m_1, m_2\}) & H_2 = \{m_1, m_2\} & H_2 = \{m_1, m_2\} \\ & H_3 = \{m_1\} & H_3 = \{m_1, m_2, m_3\} \end{array} \quad \xrightarrow{p_3, c?, \{a, b, c\}}$$

In brief, the message histories are designed as operational descriptions based on messages of the three kinds of orderings on events presented in Chapter 2. The validity of this statement is checked in the next section.

Event ordering	Message history
Total ordering (runs)	Global history
Local ordering on peers	Local histories without update at reception
Causal ordering (distributed executions)	Local histories with update at reception (sometimes called “causal histories”)

The histories have a second role: they identify messages. Since history variables increase when a message is sent and never decrease afterwards, the message history carried by each message is unique.

Model-specific Properties

The *Fully Asynchronous* and *Realisable with Synchronous Communication* models only make use of the message histories to identify messages: there is not any particular ordering property

involved at reception. It does not matter in *Fully Asynchronous* communication and there is at most one message in transit in *RSC*.

The specification of the other five communication models differ on two elements related to the ordering of events summed up in Table 2.1 on page 38:

1. The use of local histories or a global history. This depends on the ordering on send events. If the characteristic ordering property of the communication model involves a global order on the emissions, then a global history is used such as in *FIFO n-1* and *FIFO n-n*. If it involves a local order per peer, local histories are used such as in *FIFO 1-1* and *FIFO 1-n* and the provenance of messages are compared. Finally, causal ordering (*Causal* model) requires local histories with the previously described update at reception.
2. Whether or not the listened channels are taken into account to decide on which messages in transit might prevent a reception. This depends on the ordering on receive events. If the ordering property of the communication model involves a global order on the receptions such as in *FIFO n-n* and *FIFO 1-n*, the communication model shall not selectively ignore some messages in transit based on the listened channels. However, when the communication model only involves local ordering of the receptions, it is allowed to deliver a message whose history contains messages in transit on channels that are not listened to. As explained in 4.1.2 and proved in the next section, it makes sense when dealing with peers that are stable with regard to interest.

Formal Specification

Definition 24 (Specification of the Communication Models with Message Histories). *Let the set of operational messages $\mathcal{M} \triangleq \emptyset \cup \mathcal{C} \times \mathcal{P} \times 2^{\mathcal{M}}$ associated to the peers in \mathcal{P} and the channels in \mathcal{C} . The specification of the communication models are:*

$$\begin{array}{ll}
 RSC & M_{RSC} \triangleq (S_{RSC}, I_{RSC}, R_{RSC}, L) \\
 FIFO\ n-n & M_{nn} \triangleq (S_{nn}, I_{nn}, R_{nn}, L) \\
 FIFO\ 1-n & M_{1n} \triangleq (S_{1n}, I_{1n}, R_{1n}, L) \\
 FIFO\ n-1 & M_{n1} \triangleq (S_{n1}, I_{n1}, R_{n1}, L) \\
 Causal & M_c \triangleq (S_c, I_c, R_c, L) \\
 FIFO\ 1-1 & M_{11} \triangleq (S_{11}, I_{11}, R_{11}, L) \\
 Fully\ Asynchronous & M_a \triangleq (S_a, I_a, R_a, L)
 \end{array}$$

where

$$L \subseteq \left(\begin{array}{c} \text{Internal} \\ \cup \quad \mathcal{P} \times \{\text{Send}\} \times \mathcal{C} \\ \cup \quad \mathcal{P} \times \{\text{Receive}\} \times \mathcal{C} \times 2^{\mathcal{C}} \end{array} \right)$$

The state contains the network of messages in transit and global or local message histories. The network and message histories are subsets of messages ($2^{\mathcal{M}}$).

Some models have global histories:

$$\begin{array}{rcl}
 & & \begin{array}{cc} \text{Network} & \text{Global history} \end{array} \\
 S_{nn} & \subseteq & 2^{\mathcal{M}} \times 2^{\mathcal{M}} \\
 S_{n1} & \subseteq & 2^{\mathcal{M}} \times 2^{\mathcal{M}} \\
 I_{nn} = I_{n1} & = & (\emptyset, \emptyset)
 \end{array}$$

Some models have local histories (one per peer):

		Network	Local histories
S_{RSC}	\subseteq	$2^{\mathcal{M}}$	$\times (\mathcal{P} \rightarrow 2^{\mathcal{M}})$
S_{1n}	\subseteq	$2^{\mathcal{M}}$	$\times (\mathcal{P} \rightarrow 2^{\mathcal{M}})$
S_c	\subseteq	$2^{\mathcal{M}}$	$\times (\mathcal{P} \rightarrow 2^{\mathcal{M}})$
S_{11}	\subseteq	$2^{\mathcal{M}}$	$\times (\mathcal{P} \rightarrow 2^{\mathcal{M}})$
S_a	\subseteq	$2^{\mathcal{M}}$	$\times (\mathcal{P} \rightarrow 2^{\mathcal{M}})$
$I_{RSC} = I_{1n} = I_c = I_{11} = I_a$	$=$	$(\emptyset$	$, (\emptyset)_{p \in \mathcal{P}})$

The specification of the communication transitions that guarantee the different ordering policies are:

$$R_{RSC} \triangleq \left(\begin{array}{c} \left\{ \begin{array}{l} (net, (H_p)_{p \in \mathcal{P}}), \\ (p, \text{Send}, c), \\ (net', (H'_p)_{p \in \mathcal{P}}) \end{array} \right\} \left| \begin{array}{l} net = \emptyset \\ net' = net \cup \{(c, p, H_p)\} \\ H'_p = H_p \cup \{(c, p, H_p)\} \\ \forall q \in \mathcal{P} \setminus \{p\} : H'_q = H_q \end{array} \right. \right\} \quad \text{Send} \\ \cup \left\{ \begin{array}{l} (net, (H_p)_{p \in \mathcal{P}}), \\ (p, \text{Receive}, c, L), \\ (net', (H'_p)_{p \in \mathcal{P}}) \end{array} \right\} \left| \begin{array}{l} \exists (c_1, p_1, h_1) \in net : \\ c_1 = c \\ net' = net \setminus \{(c_1, p_1, h_1)\} \\ \forall q \in \mathcal{P} : H'_q = H_q \end{array} \right. \right\} \quad \text{Receive} \end{array} \right)$$

$$R_{nn} \triangleq \left(\begin{array}{c} \left\{ \begin{array}{l} (net, H), \\ (p, \text{Send}, c), \\ (net', H') \end{array} \right\} \left| \begin{array}{l} net' = net \cup \{(c, p, H)\} \\ H' = H \cup \{(c, p, H)\} \end{array} \right. \right\} \quad \text{Send} \\ \cup \left\{ \begin{array}{l} (net, H), \\ (p, \text{Receive}, c, L), \\ (net', H') \end{array} \right\} \left| \begin{array}{l} \exists (c_1, p_1, h_1) \in net : \\ c_1 = c \\ \neg \exists (c_2, p_2, h_2) \in net : \\ (c_2, p_2, h_2) \in h_1 \\ net' = net \setminus \{(c_1, p_1, h_1)\} \\ H' = H \end{array} \right. \right\} \quad \text{Receive} \end{array} \right)$$

$$R_{n1} \triangleq \left(\begin{array}{c} \left\{ \begin{array}{l} (net, H), \\ (p, \text{Send}, c), \\ (net', H') \end{array} \right\} \left| \begin{array}{l} net' = net \cup \{(c, p, H)\} \\ H' = H \cup \{(c, p, H)\} \end{array} \right. \right\} \quad \text{Send} \\ \cup \left\{ \begin{array}{l} (net, H), \\ (p, \text{Receive}, c, L), \\ (net', H') \end{array} \right\} \left| \begin{array}{l} \exists (c_1, p_1, h_1) \in net : \\ c_1 = c \\ \neg \exists (c_2, p_2, h_2) \in net : \\ c_2 \in L \\ (c_2, p_2, h_2) \in h_1 \\ net' = net \setminus \{(c_1, p_1, h_1)\} \\ H' = H \end{array} \right. \right\} \quad \text{Receive} \end{array} \right)$$

$$\begin{aligned}
R_{1n} &\triangleq \left(\begin{array}{c} \left\{ \begin{array}{c} (net, (H_p)_{p \in \mathcal{P}}), \\ (p, \text{Send}, c), \\ (net', (H'_p)_{p \in \mathcal{P}}) \end{array} \right\} \mid \begin{array}{l} net' = net \cup \{(c, p, H_p)\} \\ H'_p = H_p \cup \{(c, p, H_p)\} \\ \wedge \forall q \in \mathcal{P} \setminus \{p\} : H'_q = H_q \end{array} \end{array} \right\} \quad \text{Send} \\ \cup \left\{ \begin{array}{c} (net, (H_p)_{p \in \mathcal{P}}), \\ (p, \text{Receive}, c, L), \\ (net', (H'_p)'_{p \in \mathcal{P}}) \end{array} \right\} \mid \begin{array}{l} \exists (c_1, p_1, h_1) \in net : \\ c_1 = c \\ \neg \exists (c_2, p_2, h_2) \in net : \\ p_1 = p_2 \\ \wedge (c_2, p_2, h_2) \in h_1 \\ \wedge net' = net \setminus \{(c_1, p_1, h_1)\} \\ \wedge \forall q \in \mathcal{P} : H'_q = H_q \end{array} \end{array} \right\} \quad \text{Receive} \end{array} \right) \\
R_{11} &\triangleq \left(\begin{array}{c} \left\{ \begin{array}{c} (net, (H_p)_{p \in \mathcal{P}}), \\ (p, \text{Send}, c), \\ (net', (H'_p)_{p \in \mathcal{P}}) \end{array} \right\} \mid \begin{array}{l} net' = net \cup \{(c, p, H_p)\} \\ H'_p = H_p \cup \{(c, p, H_p)\} \\ \wedge \forall q \in \mathcal{P} \setminus \{p\} : H'_q = H_q \end{array} \end{array} \right\} \quad \text{Send} \\ \cup \left\{ \begin{array}{c} (net, (H_p)_{p \in \mathcal{P}}), \\ (p, \text{Receive}, c, L), \\ (net', (H'_p)'_{p \in \mathcal{P}}) \end{array} \right\} \mid \begin{array}{l} \exists (c_1, p_1, h_1) \in net : \\ c_1 = c \\ \neg \exists (c_2, p_2, h_2) \in net : \\ p_1 = p_2 \\ \wedge c_2 \in L \\ \wedge (c_2, p_2, h_2) \in h_1 \\ \wedge net' = net \setminus \{(c_1, p_1, h_1)\} \\ \wedge \forall q \in \mathcal{P} : H'_q = H_q \end{array} \end{array} \right\} \quad \text{Receive} \end{array} \right) \\
R_c &\triangleq \left(\begin{array}{c} \left\{ \begin{array}{c} (net, (H_p)_{p \in \mathcal{P}}), \\ (p, \text{Send}, c), \\ (net', (H'_p)_{p \in \mathcal{P}}) \end{array} \right\} \mid \begin{array}{l} net' = net \cup \{(c, p, H_p)\} \\ H'_p = H_p \cup \{(c, p, H_p)\} \\ \wedge \forall q \in \mathcal{P} \setminus \{p\} : H'_q = H_q \end{array} \end{array} \right\} \quad \text{Send} \\ \cup \left\{ \begin{array}{c} (net, (H_p)_{p \in \mathcal{P}}), \\ (p, \text{Receive}, c, L), \\ (net', (H'_p)'_{p \in \mathcal{P}}) \end{array} \right\} \mid \begin{array}{l} \exists (c_1, p_1, h_1) \in net : \\ c_1 = c \\ \neg \exists (c_2, p_2, h_2) \in net : \\ c_2 \in L \\ \wedge (c_2, p_2, h_2) \in h_1 \\ \wedge net' = net \setminus \{(c_1, p_1, h_1)\} \\ \wedge H'_p = H'_p \cup h_1 \cup \{(c_1, p_1, h_1)\} \\ \wedge \forall q \in \mathcal{P} \setminus \{p\} : H'_q = H_q \end{array} \end{array} \right\} \quad \text{Receive} \end{array} \right) \\
R_a &\triangleq \left(\begin{array}{c} \left\{ \begin{array}{c} (net, (H_p)_{p \in \mathcal{P}}), \\ (p, \text{Send}, c), \\ (net', (H'_p)_{p \in \mathcal{P}}) \end{array} \right\} \mid \begin{array}{l} net' = net \cup \{(c, p, H_p)\} \\ H'_p = H_p \cup \{(c, p, H_p)\} \\ \wedge \forall q \in \mathcal{P} \setminus \{p\} : H'_q = H_q \end{array} \end{array} \right\} \quad \text{Send} \\ \cup \left\{ \begin{array}{c} (net, (H_p)_{p \in \mathcal{P}}), \\ (p, \text{Receive}, c, L), \\ (net', (H'_p)'_{p \in \mathcal{P}}) \end{array} \right\} \mid \begin{array}{l} \exists (c_1, p_1, h_1) \in net : \\ c_1 = c \\ \wedge net' = net \setminus \{(c_1, p_1, h_1)\} \\ \wedge \forall q \in \mathcal{P} : H'_q = H_q \end{array} \end{array} \right\} \quad \text{Receive} \end{array} \right)
\end{aligned}$$

4.1.7 Specification of Capped Asynchronous Communication

It is also possible to count and/or limit the number of messages in transit, for the full network or on its projections (number of messages in transit for each channel). A counter of messages in

transit is updated during the send and receive transitions. This counter can be used to ascertain through a dedicated compatibility property if the number of messages in transit is effectively capped. A cap that prevents send transitions can be used to quickly check if a system may be realisable. Note that this implies that the emission of a message is not always enabled. In particular, the specification of the *RSC* communication models correspond to an asynchronous communication model where the bound on the number of messages in transit is 1.

4.2 Conformance to the Specifications using Distributed Executions

This section explains why the operational specifications of the communication models in Definition 24 are consistent with the characterisations using runs of events in the definitions of Chapter 2. We identify necessary conditions to prove the correctness and completeness of each communication model.

The specification of a communication model is correct when the traces of a composition of peers under that model (Definition 22) correspond to valid runs of events under this model. In some cases (namely *FIFO n-1*, *Causal*, and *FIFO 1-1*), we assume the peers in the composition are stable with regard to interest (Definition 20).

The specification of a model is complete when all the runs that are valid according to the definition from Chapter 2 can be generated in a composition of peers under this model. We actually restrict the property to runs in which all sent messages are eventually received and in the same special cases as for correctness, we assume systems with mono-receptor channels only.

Correctness and completeness assume a link between the overall product systems (compositions of peers under a communication model) and runs of events. Runs are sequences of events based on messages; the traces of the product system are sequences of states and the specification of the peers and communication model on which it is based are labelled by communication events on channels. We first describe how to formally link these two views.

Channels and messages As previously established, the message histories in the specifications of the communication models serve as unique identifiers for messages. Therefore we choose to work on runs of messages in \mathcal{M} . From now on, in this section, $\mathcal{M} \triangleq \mathcal{M}$.

Traces and runs Given a trace of the product system between a composition of peers and a communication model, the corresponding run is based on the sequence of associated transitions in the specification of the communication model we call the dual of the trace. We consider maximal traces: they are all infinite because the system stutters.

Definition 25 (Maximal Trace of a System). *Given the specification of a system (S, I, R) . A maximal trace is an infinite sequence of states $(s_i)_{i \in \mathbb{N}}$ such that $s_0 \in I$ and $\forall i \in \mathbb{N} : (s_i, s_{i+1}) \in R$. We denote $\text{traces}((S, I, R))$ the set of maximal traces of (S, I, R) :*

$$\text{traces}((S, I, R)) \triangleq \left\{ (s_i)_{i \in \mathbb{N}} \in (\mathbb{N} \rightarrow S) \mid \begin{array}{l} s_0 \in I \\ \wedge \quad \forall i \in \mathbb{N} : (s_i, s_{i+1}) \in R \end{array} \right\}$$

Definition 26 (Dual of a trace). *Let $\text{CM} \triangleq (S_{\text{CM}}, I_{\text{CM}}, R_{\text{CM}}, L_{\text{CM}})$ the specification of a communication model (Definition 24), $(P_p)_{p \in \mathcal{P}} \triangleq (S_p, I_p, R_p, L_p)_{p \in \mathcal{P}}$ the specifications of peers and (S, I, R) the specification of the composition of these peers under CM.*

$$\forall (s_i)_{i \in \mathbb{N}} \in \text{traces}((S, I, R)) :$$

dual is the least fixed point solution to:

$$\text{dual}((s_i)_{i \in \mathbb{N}}) = \text{event}(s_0, s_1) \cdot \text{dual}((s_{i+1})_{i \in \mathbb{N}})$$

where \cdot is a sequence constructor, \emptyset is a value such that $\forall \text{seq} : \emptyset \cdot \text{seq} = \text{seq}$, and event extracts the communication or internal event that happens between two states in the product system:

$$\forall s, s' \in S : \left(\begin{array}{c} \exists s_P, s'_P \in \prod_{p \in \mathcal{P}} S_p \\ \exists s_{\text{CM}} \in S_{\text{CM}} \\ \exists s'_{\text{CM}} \in S_{\text{CM}} \\ \exists \text{net}, \text{net}', h, h' \in 2^{\mathcal{M}} \end{array} : \left(\begin{array}{c} s_{\text{CM}} = (\text{net}, h) \\ s'_{\text{CM}} = (\text{net}', h') \\ s = (s_P, s_{\text{CM}}) \\ s' = (s'_P, s'_{\text{CM}}) \end{array} \right) \right) \Rightarrow$$

$$\text{event}(s, s') \triangleq \begin{cases} \emptyset & \text{if } \left(\begin{array}{c} \forall p \in \mathcal{P} : s_p = s'_p \\ \wedge \left(\begin{array}{c} s_{\text{CM}} = s'_{\text{CM}} \\ \vee (s_{\text{CM}}, \text{Internal}, s'_{\text{CM}}) \in S_{\text{CM}} \end{array} \right) \end{array} \right) \\ (\text{Internal}, p) & \text{if } \exists p \in \mathcal{P} : \left(\begin{array}{c} s_{\text{CM}} = s'_{\text{CM}} \\ \wedge (s_p, \text{Internal}, s'_p) \in R_p \\ \wedge s_p \neq s'_p \\ \wedge \forall q \in \mathcal{P} \setminus \{p\} : s_q = s'_q \end{array} \right) \\ (\text{Send}, p, m) & \text{if } \left(\begin{array}{c} \exists p \in \mathcal{P} \\ \exists c \in \mathcal{C} \\ \exists m \in \mathcal{M} \end{array} \right) : \left(\begin{array}{c} \text{send}(s, s', p, c) \\ \wedge \text{net}' \setminus \text{net} = \{m\} \end{array} \right) \\ (\text{Receive}, p, m) & \text{if } \left(\begin{array}{c} \exists p \in \mathcal{P} \\ \exists c \in \mathcal{C} \\ \exists m \in \mathcal{M} \end{array} \right) : \left(\begin{array}{c} \text{receive}(s, s', p, c) \\ \wedge \text{net} \setminus \text{net}' = \{m\} \end{array} \right) \end{cases}$$

The set of all duals of maximal traces of the system (S, I, R) is:

$$\text{segruns}((P_p)_{p \in \mathcal{P}}, \text{CM}) \triangleq \text{dual}(\text{traces}(S, I, R))$$

For every couple of consecutive states s and s' , the associated event is computed. If a message m has been removed from the network, it is a (Receive, m) event. If a message m has been added to the network, it is a (Send, m) event. Depending on which peer p may have changed, the event is localised. If the state of a peer p has changed but the state of the communication has not, it corresponds to an internal event (Internal, p). When all the peer remain in the same state, even if the state of the communication has changed, no new event is added to the sequence (\emptyset) because a change in the communication model is not an actual event in the distributed system: it cannot be localised.

The dual of a trace is a sequence of tuples containing the information that characterises the events that happen in a trace. The order in the sequence corresponds to the total ordering of events in a run. Yet, such sequences do not exactly fit the definition of runs from Chapter 2. A run is composed of totally ordered events and labelling functions that associate peers and messages. We need to bridge the gap between the operational specifications of communication models and the structural definitions in order to assert the correctness and completeness.

Definition 27 (Run of a Sequence). *Given $N \in \mathbb{N} \cup \infty$ and a sequence $(s_i)_{i \in 1..N} \in 1..N \rightarrow (\{\text{Internal}\} \times \mathcal{P}) \cup (\{\text{Send}, \text{Receive}\} \times \mathcal{P} \times \mathcal{M})$, the run associated to that sequence is $\text{run}((s_i)_{i \in 1..N}) \triangleq (E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \prec_\sigma, \text{com}, \text{peer}, \text{mes})$ such that:*

- $E = \{s_i \mid i \in 1..N\}$

- $\forall p \in \mathcal{P} : (\text{Internal}, p) \in E \Rightarrow \left(\begin{array}{l} \text{com}((\text{Internal}, p)) = \text{Internal} \\ \wedge \text{peer}((\text{Internal}, p)) = p \end{array} \right)$
- $\forall (c, p, m) \in (\{\text{Send}, \text{Receive}\} \times \mathcal{P} \times \mathcal{M}) : (c, p, m) \in E \Rightarrow \left(\begin{array}{l} \text{com}((c, p, m)) = c \\ \wedge \text{peer}((c, p, m)) = p \\ \wedge \text{mes}((c, p, m)) = m \end{array} \right)$
- $\forall p \in \mathcal{P} : \forall i, j \in 1..N : s_i \leq_p s_j \Leftrightarrow i \leq j \wedge \text{peer}(s_i) = \text{peer}(s_j) = p$
- \prec_c derives from $(\leq_p)_{p \in \mathcal{P}}$ according to the definition of a distributed execution (Definition 1).
- $\forall i, j \in 1..N : s_i \prec_\sigma s_j \Leftrightarrow i \leq j$

Let $\text{CM} \triangleq (S_{\text{CM}}, I_{\text{CM}}, R_{\text{CM}}, L_{\text{CM}})$ the specification of a communication model from Definition 24, $(P_p)_{p \in \mathcal{P}} \triangleq (S_p, I_p, R_p, L_p)_{p \in \mathcal{P}}$, the set of runs associated to the composition of the peers under this communication model is:

$$\text{runs}((P_p)_{p \in \mathcal{P}}, \text{CM}) \triangleq \text{run}(\text{seqruns}((P_p)_{p \in \mathcal{P}}, \text{CM}))$$

In the following, we do not distinguish the duals of maximal traces and their associated runs anymore. We therefore consider the sets of actual runs associated to a composition under a communication model and we compare them to the runs that define the communication models in Chapter 2. Furthermore, when the definition of the labelling functions and orders of a run are omitted, we assume the three orders on runs are all denoted \leq , \prec_c , \prec_σ and the labelling function are denoted com , peer , and mes .

4.2.1 Correctness

Theorem 28 (The Framework Generates Point-to-Point Runs). *Given $(P_p)_{p \in \mathcal{P}}$ specifications of peers.*

$$\forall M \in \{M_{RSC}, M_{nn}, M_{1n}, M_{n1}, M_c, M_{11}, M_a\} : \text{runs}((P_p)_{p \in \mathcal{P}}, M) \subseteq \text{Run}^{\text{P2P}}$$

Proof.

1. Messages are unique.
OK the histories expand when a new message is introduced and never decrease afterwards
2. A receive event is preceded by a send event.
OK a reception can only happen if the message is in the network (which is initially empty) and the message can only be added in the network when it is sent.
3. A message can only be received once.
OK a message is removed from the network (which is initially empty) when it is received.

□

In the following, the peer where an event occur is denoted $_$ when it is not relevant: this means it can be replaced by some peer in \mathcal{P} . Since the runs are all point to point runs, there is no ambiguity because messages are unique and they are sent and received at most once.

Histories and Orderings on Events

This section shows that the logical descriptions with histories realise the causal, local, and total orders on events in runs.

Theorem 29 (Order Encoding). *Histories correctly encode the orders on events (local, causal, and total). Given $(P_p)_{p \in \mathcal{P}}$ specifications of peers, $N \in \mathbb{N} \cup \infty$, $(\sigma_i)_{i \in 0..N} \in \text{runs}((P_p)_{p \in \mathcal{P}}, M)$, and two distinct messages $m_1 = (c_1, p_1, h_1)$ and $m_2 = (c_2, p_2, h_2)$ sent in the sequence. Depending on M :*

$$\begin{array}{l|l} M_{11} & (\text{Send}, _, m_1) \leq_{p_1} (\text{Send}, _, m_2) \Leftrightarrow m_1 \in h_2 \\ M_{1n} & (\text{Send}, _, m_1) \leq_{p_1} (\text{Send}, _, m_2) \Leftrightarrow m_1 \in h_2 \\ M_c & (\text{Send}, _, m_1) \prec_c (\text{Send}, _, m_2) \Leftrightarrow m_1 \in h_2 \\ M_{nn} & (\text{Send}, _, m_1) \prec_\sigma (\text{Send}, _, m_2) \Leftrightarrow m_1 \in h_2 \\ M_{n1} & (\text{Send}, _, m_1) \prec_\sigma (\text{Send}, _, m_2) \Leftrightarrow m_1 \in h_2 \end{array}$$

Proof.

1. Local ordering: M_{11} and M_{1n} (local histories $(H_p)_{p \in \mathcal{P}}$)

- \Rightarrow (hyp: $(\text{Send}, p_1, m_1) \leq_{p_1} (\text{Send}, p_2, m_2)$)

$$\exists i, j \in \mathbb{N} : \left(\begin{array}{l} (\text{Send}, p_1, m_1) = \sigma_i \\ \wedge (\text{Send}, p_2, m_2) = \sigma_j \\ \wedge p_1 = p_2 \\ \wedge i + 1 \leq j \end{array} \right)$$

$$\text{because } \left(\begin{array}{l} (\text{Send}, p_1, m_1) \leq_{p_1} (\text{Send}, p_2, m_2) \\ \wedge m_1 \neq m_2 \end{array} \right).$$

Since the histories can only grow over time, at time j , $m_1 \in H_{p_1}$ because after sending m_1 (from time $i + 1$) $m_1 \in H_{p_1}$.

QED because h_2 is H_{p_2} at time j , that is H_{p_1} at time j because $p_1 = p_2$.

- \Leftarrow (hyp: $m_1 \in h_2$)

Let j the time when m_2 is sent: $(\text{Send}, p_2, m_2) = \sigma_j$.

$p_1 = p_2$ because by construction in a peer history, all the messages are from the same peer.

$\exists i \in \mathbb{N} : i + 1 \leq j$ and $m_1 \in H_{p_1}$ at $i + 1$ but $m_1 \notin H_{p_1}$ at i because at 0, $H_{p_1} = \emptyset$.

$\sigma_i = (\text{Send}, p_1, m_1)$ is the only reason why m_1 is put in the local history H_{p_1} .

QED because $i < j$ and $p_1 = p_2$.

2. Causal ordering: M_c (causal histories $(H_p)_{p \in \mathcal{P}}$)

A history carries the causal past of a message, that is the set a messages which causally precede this message [BJ87, Bir96, KS11, Ray13].

- Send event: the current causal past of the peer is piggybacked in the sent message m , and the causal past of the next sent message from this peer will contain m .
- Receive event: the causal past of the peer becomes the union of the current causal past of the peer, of the causal past of the received message (piggybacked in the message), and the message itself. Thus, the causal past of a future message from this peer will have all the messages which causally precede it, from the same peer or from peers which have, directly or indirectly, communicated with it.

3. Total ordering: M_{n1} and M_{nn} (global history H)

- \Rightarrow (hyp: $(\text{Send}, _, m_1) \prec_\sigma (\text{Send}, _, m_2)$)
 $\exists i, j \in \mathbb{N} : \left(\begin{array}{l} (\text{Send}, _, m_1) = \sigma_i \\ \wedge (\text{Send}, _, m_2) = \sigma_j \\ \wedge i + 1 \leq j \end{array} \right)$
because $\left(\begin{array}{l} (\text{Send}, _, m_1) \prec_\sigma (\text{Send}, _, m_2) \\ \wedge m_1 \neq m_2 \end{array} \right)$.

Since the histories can only grow over time, at time j , $m_1 \in H$ because after sending m_1 (from time $i + 1$) $m_1 \in H$.

QED because h_2 is H at time j .

- \Leftarrow (hyp: $m_1 \in h_2$)

Let j the time when m_2 is sent: $(\text{Send}, _, m_2) = \sigma_j$.

$\exists i \in \mathbb{N} : i + 1 \leq n$ and $m_1 \in H$ at $i + 1$ but $m_1 \notin H$ at i because at 0, $H = \emptyset$.

$\sigma_i = (\text{Send}, _, m_1)$ is the only reason why m_1 is put in the global history H .

QED because $i < j$.

□

We now show that the specifications of the communication models with message histories conform to the structural definitions of the communication models in Chapter 2. Some specifications of communication models are only correct when involved in compositions of peers that are stable with regard to interest (Definition 20).

Theorem 30 (Correctness of the Framework). *Given $(P_p)_{p \in \mathcal{P}}$ specifications of peers.*

- $\text{runs}((P_p)_{p \in \mathcal{P}}, M_{RSC}) \subseteq \text{Run}_{RSC}^{\text{P2P}}$
- $\text{runs}((P_p)_{p \in \mathcal{P}}, M_{nn}) \subseteq \text{Run}_{nn}^{\text{P2P}}$
- $\forall p \in \mathcal{P} : p \text{ is stable with regard to interest} \Rightarrow \text{runs}((P_p)_{p \in \mathcal{P}}, M_{n1}) \subseteq \text{Run}_{n1}^{\text{P2P}}$
- $\text{runs}((P_p)_{p \in \mathcal{P}}, M_{1n}) \subseteq \text{Run}_{1n}^{\text{P2P}}$
- $\forall p \in \mathcal{P} : p \text{ is stable with regard to interest} \Rightarrow \text{runs}((P_p)_{p \in \mathcal{P}}, M_c) \subseteq \text{Run}_c^{\text{P2P}}$
- $\forall p \in \mathcal{P} : p \text{ is stable with regard to interest} \Rightarrow \text{runs}((P_p)_{p \in \mathcal{P}}, M_{11}) \subseteq \text{Run}_{11}^{\text{P2P}}$
- $\text{runs}((P_p)_{p \in \mathcal{P}}, M_a) \subseteq \text{Run}^{\text{P2P}}$

Proof. **Correctness of RSC** given the condition on the send and receive transitions of M_{RSC} , a send transition can only occur if $net = \emptyset$ and a receive transition can only occur if $net \neq \emptyset$. A send transition ensures that $net \neq \emptyset$ in the next state and a receive transition ensures that $net = \emptyset$ in the next state. Thus, send and receive events (with interleaved internal events) alternate in runs of $\text{runs}(M_{RSC})$ which is conform to $\text{Run}_{RSC}^{\text{P2P}}$.

Correctness of $FIFO$ n - n Let $\sigma \in \text{runs}((P_p)_{p \in \mathcal{P}}, M_{nn})$.

$$\begin{array}{lcl} \text{Let } m_1 \triangleq (c_1, p_1, h_1) \text{ and } m_2 \triangleq (c_2, p_2, h_2) \text{ such that:} & & (\text{Send}, _, m_1) \prec_\sigma (\text{Send}, _, m_2) \\ & \wedge & (\text{Receive}, _, m_1) \in \sigma \\ & \wedge & (\text{Receive}, _, m_2) \in \sigma \end{array}$$

$m_1 \in h_2$ by Theorem 29.

Assume that $(\text{Receive}, _, m_2) \prec_\sigma (\text{Receive}, _, m_1)$.

When $(\text{Receive}, _, m_2)$ happens, $m_1 \in \text{net}$ because $m_1 \neq m_2$, $(\text{Send}, _, m_1) \prec_\sigma (\text{Send}, _, m_2)$ and $(\text{Receive}, _, m_2) \prec_\sigma (\text{Receive}, _, m_1)$.

Contradiction. The specification of M_{nn} requires that $m_1 \notin \text{net}$ because $m_1 \in h_2$.

QED $\sigma \in \text{Run}_{nn}^{\text{P2P}}$ because $(\text{Receive}, _, m_1) \prec_\sigma (\text{Receive}, _, m_2)$.

Correctness of FIFO n-1 Let $\sigma \in \text{runs}((P_p)_{p \in \mathcal{P}}, M_{n1})$.

Let $p \in \mathcal{P}$.

Let $m_1 \triangleq (c_1, p_1, h_1)$ and $m_2 \triangleq (c_2, p_2, h_2)$ such that:

$$\begin{aligned} & (\text{Send}, _, m_1) \prec_\sigma (\text{Send}, _, m_2) \\ & \wedge (\text{Receive}, p, m_1) \in \sigma \\ & \wedge (\text{Receive}, p, m_2) \in \sigma \end{aligned}$$

$m_1 \in h_2$ by Theorem 29.

Assume that $(\text{Receive}, p, m_2) \prec_\sigma (\text{Receive}, p, m_1)$.

When $(\text{Receive}, p, m_2)$ happens, $m_1 \in \text{net}$ because $m_1 \neq m_2$, $(\text{Send}, _, m_1) \prec_\sigma (\text{Send}, _, m_2)$ and $(\text{Receive}, p, m_2) \prec_\sigma (\text{Receive}, p, m_1)$.

In order for $(\text{Receive}, p, m_2)$ to happen, p is not interested by c_1 . This comes from the specification of M_{n1} because $m_1 \in h_2$.

In order for $(\text{Receive}, p, m_1)$ to happen, p is interested by c_1 . This comes from the specification of M_{n1} .

Contradiction. p is *stable with regard to interest* but at a given time it is not interested in c_1 and later is.

QED $\sigma \in \text{Run}_{n1}^{\text{P2P}}$ because $(\text{Receive}, p, m_1) \prec_\sigma (\text{Receive}, p, m_2)$.

Correctness of FIFO 1-n Let $\sigma \in \text{runs}((P_p)_{p \in \mathcal{P}}, M_{1n})$.

Let $m_1 \triangleq (c_1, p_1, h_1)$ and $m_2 \triangleq (c_2, p_2, h_2)$ such that:

$$\begin{aligned} & (\text{Send}, p_1, m_1) \prec_c (\text{Send}, p_2, m_2) \\ & \wedge p_1 = p_2 \\ & \wedge (\text{Receive}, _, m_1) \in \sigma \\ & \wedge (\text{Receive}, _, m_2) \in \sigma \end{aligned}$$

$m_1 \in h_2$ by Theorem 29 because $(\text{Send}, p_1, m_1) \leq_{p_1} (\text{Send}, p_2, m_2)$.

Assume that $(\text{Receive}, _, m_2) \prec_\sigma (\text{Receive}, _, m_1)$.

When $(\text{Receive}, _, m_2)$ happens, $m_1 \in \text{net}$ because $m_1 \neq m_2$, $(\text{Send}, p_1, m_1) \prec_\sigma (\text{Send}, p_2, m_2)$ and $(\text{Receive}, _, m_2) \prec_\sigma (\text{Receive}, _, m_1)$.

Contradiction. The specification of M_{1n} requires that $m_1 \notin \text{net}$ because $m_1 \in h_2$ and $p_1 = p_2$.

QED $\sigma \in \text{Run}_{1n}^{\text{P2P}}$ because $(\text{Receive}, _, m_1) \prec_\sigma (\text{Receive}, _, m_2)$.

Correctness of Causal Let $\sigma \in \text{runs}((P_p)_{p \in \mathcal{P}}, M_c)$.

Let $p \in \mathcal{P}$.

Let $m_1 \triangleq (c_1, p_1, h_1)$ and $m_2 \triangleq (c_2, p_2, h_2)$ such that:

$$\begin{aligned} & (\text{Send}, _, m_1) \prec_c (\text{Send}, _, m_2) \\ & \wedge (\text{Receive}, p, m_1) \in \sigma \\ & \wedge (\text{Receive}, p, m_2) \in \sigma \end{aligned}$$

$m_1 \in h_2$ by Theorem 29.

Assume that $(\text{Receive}, p, m_2) \prec_\sigma (\text{Receive}, p, m_1)$.

When (Receive, p, m_2) happens, $m_1 \in \text{net}$ because $m_1 \neq m_2$, $(\text{Send}, _, m_1) \prec_c (\text{Send}, _, m_2)$ and $(\text{Receive}, p, m_2) \prec_\sigma (\text{Receive}, p, m_1)$.

In order for (Receive, p, m_2) to happen, p is not interested by c_1 . This comes from the specification of M_c because $m_1 \in h_2$.

In order for (Receive, p, m_1) to happen, p is interested by c_1 . This comes from the specification of M_c .

Contradiction. p is *stable with regard to interest* but at a given time it is not interested in c_1 and later is.

QED $\sigma \in \text{Run}_c^{\text{P2P}}$ because $(\text{Receive}, p, m_1) \prec_\sigma (\text{Receive}, p, m_2)$.

Correctness of FIFO 1-1 Let $\sigma \in \text{runs}((P_p)_{p \in \mathcal{P}}, M_{11})$.

Let $p \in \mathcal{P}$.

	$(\text{Send}, p_1, m_1) \prec_c (\text{Send}, p_2, m_2)$
Let $m_1 \triangleq (c_1, p_1, h_1)$ and $m_2 \triangleq (c_2, p_2, h_2)$ such that:	$\wedge \quad p_1 = p_2$
	$\wedge \quad (\text{Receive}, p, m_1) \in \sigma$
	$\wedge \quad (\text{Receive}, p, m_2) \in \sigma$

$m_1 \in h_2$ by Theorem 29.

Assume that $(\text{Receive}, p, m_2) \prec_\sigma (\text{Receive}, p, m_1)$.

When (Receive, p, m_2) happens, $m_1 \in \text{net}$ because $m_1 \neq m_2$, $(\text{Send}, _, m_1) \prec_c (\text{Send}, _, m_2)$ and $(\text{Receive}, p, m_2) \prec_\sigma (\text{Receive}, p, m_1)$.

In order for (Receive, p, m_2) to happen, p is not interested by c_1 . This comes from the specification of M_{11} because $m_1 \in h_2$ and $p_1 = p_2$.

In order for (Receive, p, m_1) to happen, p is interested by c_1 . This comes from the specification of M_{11} .

Contradiction. p is *stable with regard to interest* but at a given time it is not interested in c_1 and later is.

QED $\sigma \in \text{Run}_{11}^{\text{P2P}}$ because $(\text{Receive}, p, m_1) \prec_\sigma (\text{Receive}, p, m_2)$.

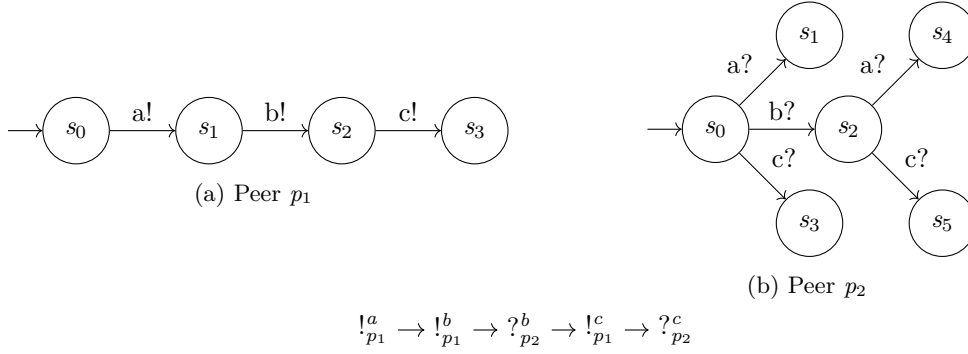
Correctness of Fully Asynchronous The framework generates point-to-point runs (Theorem 28).

□

4.2.2 Completeness

The specification of a communication model is complete when it can generate any run that is valid according to the definitions in Chapter 2. The runs that are generated by the *Fully Asynchronous* communication model (whose specification states that sending a message or receiving a message in transit is always allowed) constitute the reference pool of runs. The runs the other communication models generate are in this pool of *Fully Asynchronous*.

First, we restrict the study of the completeness to a subset of these runs we call “fair”. In fair runs, all the messages in transit are eventually received. When a system deadlocks (termination, unexpected reception, communication deadlock), some messages may be left in transit and the resulting finite run is unfair. Yet, these are common cases it is not acceptable to ignore. Therefore, we prove that completeness also holds for runs that can be extended to fair runs: if an extension can be generated by the communication model, any prefix can, including the finite unfair run of interest. We then provide an actual way to extend any finite unfair run



(c) Problematic Unfair Run. The send (resp. receive) event of a message on channel c by peer p is denoted $!_p^c$ (resp. $?_p^c$).

Figure 4.6: Two peers and an example run that illustrate the issue with unfair runs under communication models such as *FIFO 1-1*.

into a fair run taking the communication properties into account. For some of the communication models (namely *FIFO 1-1*, *Causal*, and *FIFO n-1*), we prove the completeness for single receptor systems only. In such systems, at most one peer may listen to a given channel.

Once again, this section considers runs as sequences of events of the form (Send, p, m) , $(\text{Receive}, p, m)$, and $(\text{Internal}, p)$ where p is a peer and m a message. See Section 4.2 for more details. Given σ a run and e an event, $e \in \sigma$ means that e is an element of the sequence of events σ .

Definition 31 (Fair Runs). *A run is fair when all the sent messages are eventually received.*

$$\forall \sigma \in \text{Run}^{\text{P2P}} : \text{fair}(\sigma) \triangleq \left(\forall m \in \mathcal{M} : \forall p \in \mathcal{P} : (\text{Send}, p, m) \in \sigma \Rightarrow \left(\begin{array}{l} \exists q \in \mathcal{P} : \\ (\text{Receive}, q, m) \in \sigma \end{array} \right) \right)$$

Let us consider the two peers in Figure 4.6a and 4.6b. The run in Figure 4.6c can be generated by the two peers under the *Fully Asynchronous* communication model: it is in runs $((p_1, p_2), M_a)$. This run is a valid *FIFO 1-1* run according to Definition 10 in Chapter 2 (page 27): it is in Run_{11} . However, this run is not expected to be generated by the framework, it is not in runs $((p_1, p_2), M_{11})$, because of the oracle issue described at the beginning of this Chapter and in Figure 4.1. In state s_0 , p_2 listens to channels a , b , and c . According to the specification M_{11} of the *FIFO 1-1* communication model in the framework, once a message on channel a has been sent, p_2 has to receive this message before any other. The reception of the message on b is impossible and the run cannot be generated. The run is unfair: the message on a is put aside. In an unfair run, any troublesome message can be ignored, thus any send behaviour could be turned into a valid receive behaviour. Yet, the framework aims at checking if a composition of peers is compatible, that is, if their send behaviour is compatible with their receive behaviour. For now, we prove a loose version of completeness with fair runs only.

Definition 32 (Single Receptor Composition). *In a single receptor system, the channels are restricted to one receiver. At most one peer may listen to a channel. We first overload Definition 19 of the listened channels on peers. Given $P \triangleq (S, I, R, L)$ the specification of a peer, the listened channels of P is the set of channels in possible receptions in P :*

$$\text{LC}(P) \triangleq \bigcup_{s \in S} \text{LC}_P(s)$$

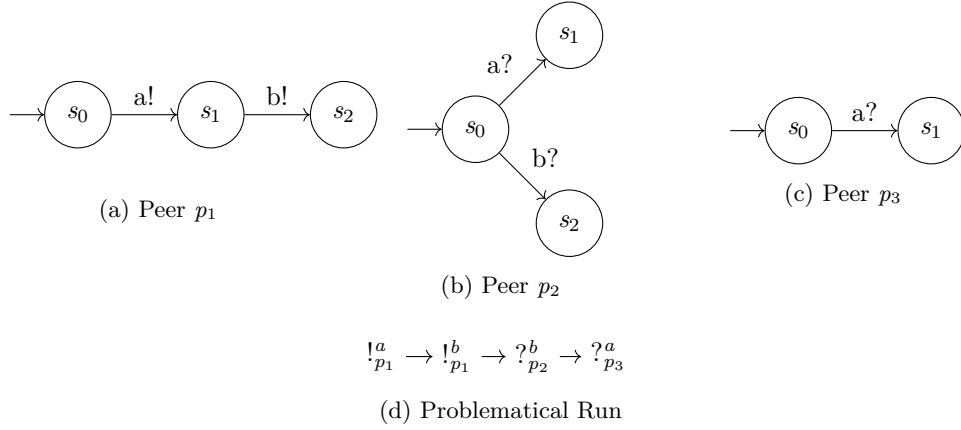


Figure 4.7: Three peers and an example run that illustrate the issue with non single receptor systems under communication models such as *FIFO 1-1*.

Given $N \in \mathbb{N}$ and $(P_p)_{p \in 1..N}$ specifications of peers, their composition is a single receptor composition when:

$$\text{SingleRecep}((P_p)_{p \in 1..N}) \triangleq \forall i, j \in 1..N : i \neq j \Rightarrow \text{LC}(P_i) \cap \text{LC}(P_j) = \emptyset$$

Let us consider the three peers in Figure 4.7a, 4.7b, and 4.7c. The run in Figure 4.7d can be generated by the two peers under the *Fully Asynchronous* communication model, it is a valid *FIFO 1-1* run, and it is fair. Completeness should then guarantee that this run can also be generated by the same peers under the *FIFO 1-1* communication model: it should be in runs $((p_1, p_2, p_3), M_{11})$. The framework cannot actually generate it because the message on channel a blocks the reception of the message on channel b by peer p_2 . There is no way to know that, *in the future*, the message on channel a will be received by another peer and allow the reception of the message on b *now*. The problem occurs because both p_2 and p_3 are listening to the same channel a . The completeness of *FIFO 1-1*, *Causal*, and *FIFO n-1*, only holds for single receptor systems.

Lemma 33 (Completeness of the Models with Fair Runs). *Given P a composition of peers.*

$$\begin{array}{llll} \sigma \in \text{Run} & \wedge \text{fair}(\sigma) & \Rightarrow & \sigma \in \text{Runs}(P, M_a) \\ \sigma \in \text{Run}_{11} & \wedge \text{fair}(\sigma) \wedge \text{SingleRecep}(P) & \Rightarrow & \sigma \in \text{Runs}(P, M_{11}) \\ \sigma \in \text{Run}_c & \wedge \text{fair}(\sigma) \wedge \text{SingleRecep}(P) & \Rightarrow & \sigma \in \text{Runs}(P, M_c) \\ \forall \sigma \in \text{Runs}(P, M_a) : & \sigma \in \text{Run}_{1n} \wedge \text{fair}(\sigma) & \Rightarrow & \sigma \in \text{Runs}(P, M_{1n}) \\ & \sigma \in \text{Run}_{n1} \wedge \text{fair}(\sigma) \wedge \text{SingleRecep}(P) & \Rightarrow & \sigma \in \text{Runs}(P, M_{n1}) \\ & \sigma \in \text{Run}_{nn} \wedge \text{fair}(\sigma) & \Rightarrow & \sigma \in \text{Runs}(P, M_{nn}) \\ & \sigma \in \text{Run}_{RSC} \wedge \text{fair}(\sigma) & \Rightarrow & \sigma \in \text{Runs}(P, M_{RSC}) \end{array}$$

Proof. Let P a composition of peers and $\sigma \in \text{Runs}(P, M_a)$.

Case *Fully Asynchronous* Trivial.

Case *RSC*

σ alternates send and receive events because $\sigma \in \text{Run}_{RSC}$ and $\text{fair}(\sigma)$ (if σ were not fair, messages could be sent and never received).

$|net| \leq 1$ in every state of the corresponding trace because of the alternation and initially $net = \emptyset$.

- If a receive transition is enabled with M_a , it is enabled in the same state with M_{RSC} (same action).
- If a send transition is enabled with M_a , the network is empty because in the next state (as in every state), $|net| \leq 1$, hence it is enabled in M_{RSC} .
- Internal transitions are always enabled.

Other communication models $*$ where $*$ may be $1l$, c , $1n$, $n1$, or nn .

Assume $\sigma \in \text{Run}_*$ and $\text{fair}(\sigma)$ but $\sigma \notin \text{Runs}(P, M_*)$

By definition of $\text{Runs}(P, M_a)$ (Definitions 25, 26, and 27), σ is the dual of a trace generated by the composition P under M_a .

P under M_* does not generate more transitions because M_* refines M_a . This result is simple and detailed both in Chapter 6 (mechanisation with Event-B), [CHMQ16] (mechanisation with TLA^+ and the TLA^+ Proof System), and [CHQ16] (mechanisation with Why3).

Thus, there is a state in which a transition with a given label is enabled in the composition P with M_a but not in P with M_* . Consider this state and transition:

1. Cases Internal and Send:

Contradiction. Internal transitions are always enabled in M_* . Send transitions too (remember that M_* is not M_{RSC}).

2. Case Receive: Let $m_2 \triangleq (c_2, p_2, h_2)$ the message that cannot be received in M_* by a peer whose specification is denoted RP (Receiving Peer) in the following.

A message $m_1 \triangleq (c_1, p_1, h_1)$ blocks the reception:

- It is in transit : $m_1 \in net$.
- It precedes m_2 : $m_1 \in h_2$.

m_1 is delivered at some point because $\text{fair}(\sigma)$.

m_1 is delivered after m_2 because $m_1 \in net$.

Therefore, $(\text{Receive}, _, m_2) \prec_\sigma (\text{Receive}, _, m_1)$.

- (a) Case *FIFO n-n* ($*$ = nn):

$(\text{Send}, _, m_1) \prec_\sigma (\text{Send}, _, m_2)$ by Theorem 29 because $m_1 \in h_2$.

Contradiction. $(\text{Receive}, _, m_1) \prec_\sigma (\text{Receive}, _, m_2)$ by Definition 15 (page 36) of *FIFO n-n*.

- (b) Case *FIFO 1-n* ($*$ = $1n$):

$(\text{Send}, _, m_1) \leq_\sigma (\text{Send}, _, m_2)$ by Theorem 29 because $m_1 \in h_2$.

Contradiction. $(\text{Receive}, _, m_1) \prec_\sigma (\text{Receive}, _, m_2)$ by Definition 13 (page 31) of *FIFO 1-n*.

- (c) Case *FIFO n-1* ($*$ = $n1$):

Assume P is a Single Receptor Composition: $\text{SingleRecep}(P)$.

- i. Case $\text{peer}((\text{Receive}, _, m_1)) = \text{peer}((\text{Receive}, _, m_2))$:

$(\text{Send}, _, m_1) \prec_\sigma (\text{Send}, _, m_2)$ by Theorem 29 because $m_1 \in h_2$.

Contradiction. $(\text{Receive}, _, m_1) \prec_\sigma (\text{Receive}, _, m_2)$ by Definition 12 (page 31) of *FIFO n-1*.

- ii. Case $\text{peer}((\text{Receive}, _, m_1)) \neq \text{peer}((\text{Receive}, _, m_2))$:
 Let RP_1 the specification of the other peer that received m_1 .
 $c_1 \in \text{LC}(RP_1)$
 $c_1 \in \text{LC}(RP)$ by Definition 24 of M_{n1} because m_1 blocks the reception of m_2 .
Contradiction. P is not a Single Receptor Composition because $RP_1 \neq RP$.
- (d) Case *Causal* ($* = c$):
Assume P is a Single Receptor Composition: $\text{SingleRecep}(P)$.
 - i. Case $\text{peer}((\text{Receive}, _, m_1)) = \text{peer}((\text{Receive}, _, m_2))$:
 $(\text{Send}, _, m_1) \prec_c (\text{Send}, _, m_2)$ by Theorem 29 because $m_1 \in h_2$.
Contradiction. $(\text{Receive}, _, m_1) \prec_\sigma (\text{Receive}, _, m_2)$ by Definition 11 (page 28) of *Causal*.
 - ii. Case $\text{peer}((\text{Receive}, _, m_1)) \neq \text{peer}((\text{Receive}, _, m_2))$:
 Let RP_1 the specification of the other peer that received m_1 .
 $c_1 \in \text{LC}(RP_1)$
 $c_1 \in \text{LC}(RP)$ by Definition 24 of M_c because m_1 blocks the reception of m_2 .
Contradiction. P is not a Single Receptor Composition because $RP_1 \neq RP$.
- (e) Case *FIFO 1-1* ($* = 11$):
Assume P is a Single Receptor Composition: $\text{SingleRecep}(P)$.
 - i. Case $\text{peer}((\text{Receive}, _, m_1)) = \text{peer}((\text{Receive}, _, m_2))$:
 $(\text{Send}, _, m_1) \leq (\text{Send}, _, m_2)$ by Theorem 29 because $m_1 \in h_2$.
Contradiction. $(\text{Receive}, _, m_1) \prec_\sigma (\text{Receive}, _, m_2)$ by Definition 10 (page 27) of *FIFO 1-1*.
 - ii. Case $\text{peer}((\text{Receive}, _, m_1)) \neq \text{peer}((\text{Receive}, _, m_2))$:
 Let RP_1 the specification of the other peer that received m_1 .
 $c_1 \in \text{LC}(RP_1)$
 $c_1 \in \text{LC}(RP)$ by Definition 24 of M_{11} because m_1 blocks the reception of m_2 .
Contradiction. P is not a Single Receptor Composition because $RP_1 \neq RP$.

□

As explained earlier, restricting completeness to fair runs puts aside many cases of interest: in particular termination, faulty receptions, or communication deadlocks. Consider the composition of p_1 , p_2 , and p_3 specified in Figures 4.8a, 4.8b, and 4.8c under the *FIFO 1-1* communication model. In order to ensure the three peers will always reach the terminal state 0 (termination compatibility property), *Causal* communication is necessary. In our case, the run in Figure 4.8d is interesting because it may be generated under *FIFO 1-1* and corresponds to a violation of the compatibility property. Unfortunately, it is not fair, thus, it does not fall under the scope of Lemma 33. Yet, it is different from the example we have seen earlier because there is a fair extension of that run that can be generated under the *FIFO 1-1* communication model. In the current example, the message on channel a can be received after the message on channel c because these messages have been sent by different peers: they are unrelated when it comes to *FIFO 1-1* communication. In the previous example, there was no way to extend the unfair run without violating the ordering policy of the communication model. Additional receptions have to be added to the specifications of the peers in order to empty the network from the last state of the system. The simplest way to do this consists in choosing a peer in the composition and introducing new loop reception transitions in the last visited state for every channel. This cannot apply to *FIFO 1-1*, *Causal*, and *FIFO n-1* which require the composition is single receptor. In these cases the receptions cannot be introduced in an arbitrary peer: each channel is mapped to its corresponding recipient and the new reception transitions are added to the last visited state

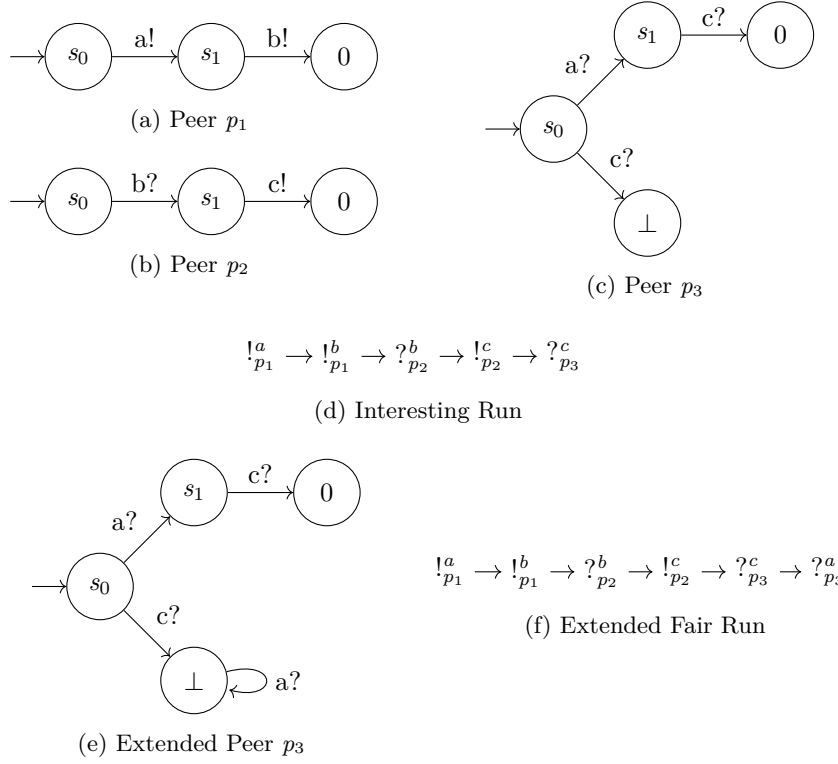


Figure 4.8: Three peers and an example run with *FIFO 1-1* that illustrate the interest of the runs that are eventually fair.

in the corresponding peers. In the current example *FIFO 1-1*, a single receptor composition is needed, thus, the reception transition on a is added in p_3 , the peer that listens to a , in the last visited state \perp , hence the specification in Figure 4.8e and the extended run it allows to generate in Figure 4.8f. The interesting unfair run is a prefix of this fair run and until the extended part begins, it is generated by a composition that fits the initial one exactly. From then, it is safe to state that the interesting run *can* be generated by the composition of peer under the *FIFO 1-1* communication model. We say such runs are “eventually fair” and we prove completeness of the models still holds.

Definition 34 (Default Destination of a Channel). *Given a composition of peers $(P_p)_{p \in 1..N}$ and a channel c in \mathcal{C} :*

$$\text{dest}((P_p)_{p \in 1..N}, c) \triangleq \begin{cases} P_i \text{ such that } c \in \text{LC}(P_i) \text{ if } \left(\begin{array}{l} \text{SingleRecep}((P_p)_{p \in 1..N}) \\ \wedge \quad c \in \text{LC}(\{P_p \mid p \in 1..N\}) \end{array} \right) \\ \text{any } P_i \text{ such that } i \in 1..N \text{ otherwise} \end{cases}$$

The default destination of a channel is an arbitrary peer in the composition unless it is a single receptor composition containing a peer that listens to this channel.

Definition 35 (Eventually Fair Runs). *An infinite run is eventually fair when it is fair. A finite run is eventually fair when it can be extended into a fair execution with respect to a communication model. Given a composition of peers $(P_p)_{p \in 1..N}$, the specification of communication model M_* where $*$ may be a , $1l$, c , $1n$, $n1$, nn , or RSC , and a run σ in $\text{Runs}((P_p)_{p \in 1..N}, M_*)$:*

- If σ is infinite: $\Diamond\text{fair}((P_p)_{p \in 1..N}, M_*, \sigma) \triangleq \text{fair}(\sigma)$
- If σ is finite

$$\Diamond\text{fair}((P_p)_{p \in 1..N}, M_*, \sigma) \triangleq \exists \sigma' : \left(\begin{array}{l} \forall e \in \sigma' : \text{com}(e) = \text{Receive} \\ \wedge \sigma \cdot \sigma' \in \text{Run}_* \\ \wedge \text{fair}(\sigma \cdot \sigma') \\ \wedge \left(\begin{array}{l} \forall e \in \sigma' : \\ \exists c \in \mathcal{C} : \\ \left(\begin{array}{l} \text{mes}(e) = (c, _, _) \\ \wedge \text{peer}(e) = \text{dest}((P_p)_{p \in 1..N}, c) \end{array} \right) \end{array} \right) \end{array} \right)$$

Theorem 36 (Completeness of the Models). *Given P a composition of peers:*

$$\forall \sigma \in \text{Runs}(P, M_a) :$$

$$\begin{array}{llll} \sigma \in \text{Run} & \wedge \Diamond\text{fair}(P, M_a, \sigma) & \Rightarrow & \sigma \in \text{Runs}(P, M_a) \\ \sigma \in \text{Run}_{11} & \wedge \Diamond\text{fair}(P, M_{11}, \sigma) \wedge \text{SingleRecep}(P) & \Rightarrow & \sigma \in \text{Runs}(P, M_{11}) \\ \sigma \in \text{Run}_c & \wedge \Diamond\text{fair}(P, M_c, \sigma) \wedge \text{SingleRecep}(P) & \Rightarrow & \sigma \in \text{Runs}(P, M_c) \\ \sigma \in \text{Run}_{1n} & \wedge \Diamond\text{fair}(P, M_{1n}, \sigma) & \Rightarrow & \sigma \in \text{Runs}(P, M_{1n}) \\ \sigma \in \text{Run}_{n1} & \wedge \Diamond\text{fair}(P, M_{n1}, \sigma) \wedge \text{SingleRecep}(P) & \Rightarrow & \sigma \in \text{Runs}(P, M_{n1}) \\ \sigma \in \text{Run}_{nn} & \wedge \Diamond\text{fair}(P, M_{nn}, \sigma) & \Rightarrow & \sigma \in \text{Runs}(P, M_{nn}) \\ \sigma \in \text{Run}_{RSC} & \wedge \Diamond\text{fair}(P, M_{RSC}, \sigma) & \Rightarrow & \sigma \in \text{Runs}(P, M_{RSC}) \end{array}$$

Proof. Let $N \in \mathbb{N}$, $(P_p)_{p \in 1..N}$ a composition, and $\sigma \in \text{Runs}((P_p)_{p \in 1..N}, M_a)$ such that $\sigma \in \text{Run}_*$ and $\Diamond\text{fair}(P, M_*, \sigma)$ where $*$ can be 11, c , $1n$, $n1$, nn , RSC , or a (in that case $\text{Run}_a \triangleq \text{Run}$).

1. Case σ infinite:

$\text{fair}(\sigma)$ by Definition 35.

QED by Lemma 33.

2. Case σ finite:

Let t the trace of the system from which σ is derived. Let $s = (s_{\text{CM}}, (s_p)_{p \in 1..N})$ the last state in t before the infinite stuttering. This state exists by Definition 26 because σ is finite.

Take σ' that extends σ into a fair run according to Definition 35. It exists because $\Diamond\text{fair}(P, M_*, \sigma)$.

$\forall p \in 1..N : \exists S_p, I_p, R_p, L_p : P_p = (S_p, I_p, R_p, L_p)$ by Definition 18.

Let $(R'_p)_{p \in 1..N} \triangleq p \in 1..N \mapsto R_p \cup \{(s_p, (\text{Receive}, c), s_p) \mid c \in \mathcal{C} \wedge P_p = \text{dest}((P_p)_{p \in 1..N}, c)\}$

Let $(P'_p)_{p \in 1..N} \triangleq (S_p, I_p, R'_p, L_p)$ a new composition of peers.

Then, $\sigma \cdot \sigma' \in \text{Runs}((P'_p)_{p \in 1..N}, M_a)$ because the new system accepts σ (the original transitions) extended by σ' (the new transitions).

$\sigma \cdot \sigma' \in \text{Run}_*$ and $\text{fair}(\sigma \cdot \sigma')$ by Definition 35.

$\sigma \cdot \sigma' \in \text{Runs}((P'_p)_{p \in 1..N}, M_*)$ by Lemma 33.

$\sigma \in \text{Runs}((P'_p)_{p \in 1..N}, M_*)$ because σ is a prefix of $\sigma \cdot \sigma'$.

$\sigma \in \text{Runs}((P_p)_{p \in 1..N}, M_*)$ since the $(P'_p)_{p \in 1..N}$ do not introduce any transition used to generate σ . **QED**.

□

4.3 Conclusion

This chapter provides an operational point of view of asynchronous communication that completes the study of the communication models described in Chapter 2 and lays down theoretical bases for certified mechanised compatibility checking of communicating peers.

We present a framework to check whether or not compositions of peers comply to good-behaviour temporal properties depending on the asynchronous communication model in use. A uniform set of operational specifications for the seven communication models is presented. Yet, they remain abstract enough not to preclude implementations. The specifications all rely on the concept of message histories that allows to establish strong generic links with the logical definitions of the ordering policies. Thus, the conformance of the framework with respect to the actual event ordering properties is established: correctness and completeness of the communication models are proven keeping the objectives of the compatibility checking in mind.

The next chapter presents an extensive mechanisation that relies on model-checking and the TLA⁺ specification language.

Chapter 5

A Mechanisation of Compatibility Checking with TLA⁺

This chapter presents a mechanisation of the framework described in Chapter 4. The notions of peers and communication models correspond to TLA⁺ specifications that are structured into independent modules and may be changed on-the-fly individually. Together, they form a system that is model-checked, with TLC (the TLA⁺ model checker), against the compatibility criteria expressed as temporal properties. To ease that process, additional user-friendly automations are proposed: they help to generate specifications of peers and their associated TLA⁺ module. The notion of composite communication model is also introduced: it makes it possible to construct a complex communication model from instances of other communication models. Each one of the building blocks correspond to a group of channels which means different parts of the system may be associated to different ordering properties. This allows to fine tune the communication properties for each practical case. The proposed mechanisation offers a ready-to-use and fully automated toolchain but there is room for easy customisation and extensibility at every step of the process. The first section provides the necessary background for the TLA⁺ Specification Language and tools. The second section presents the structure of the TLA⁺ modules involved in compatibility checking, the next section provides user-friendly automations to help the specification of compositions and communication models. Eventually the last section illustrates the mechanised framework with examples and offers insights on the model checking process and performance.

5.1 The TLA⁺ Specification Language

TLA⁺ [Lam02] is a formal specification language based on untyped Zermelo-Fraenkel set theory for specifying data structures, and on the temporal logic of actions (TLA) for specifying dynamic behaviours. Expressions rely on standard first-order logic, set operators, and several arithmetic modules. Hilbert's choice operator, written as $\text{CHOOSE } x \in S : p$, deterministically picks an arbitrary value in S which satisfies p , provided such a value exists (its value is undefined otherwise).

Functions are primitive objects in TLA⁺, and tuples are a particular kind of function. The application of function f to an expression e is written as $f[e]$. The set of functions whose domain is X and whose co-domain is a subset of Y is written as $[X \rightarrow Y]$. The expression $\text{DOMAIN } f$ is the domain of the function f . The expression $[x \in X \mapsto e]$ denotes the function with domain

X that maps any $x \in X$ to e . The notation $[f \text{ EXCEPT } ![e_1] = e_2]$ is a function which is equal to the function f except at point e_1 , provided that $e_1 \in \text{DOMAIN } f$, where its value is e_2 . Tuples (a.k.a sequences) are functions with domain $1..n, n \in \text{Nat}$. Tuples are written $\langle a_1, a_2, a_3 \rangle$. $\langle \rangle$ is the empty sequence.

Modules are used to structure complex specifications. A module contains constant declarations, variable declarations, and definitions. A module can *extend* other modules, importing all their declarations and definitions. A module can also be an *instantiation* of another module. The module $MI \triangleq \text{INSTANCE } M \text{ WITH } q_1 \leftarrow e_1, q_2 \leftarrow e_2 \dots$ is an instantiation of module M , where each symbol q_i is replaced by e_i (q_i are identifiers specifying constants or variables of module M , and e_i are expressions). Then $MI!x$ references the symbol x of the instantiated module.

Other than constant and variable declarations, a module contains definitions in the form $Op(arg_1, \dots, arg_n) \triangleq exp$. This defines the symbol Op such that $Op(e_1, \dots, e_n)$ equals exp , where each arg_i is replaced by e_i . In case of no argument, it is written as $Op \triangleq e$. A definition is just an abbreviation or syntactic sugar for an expression, and never changes its meaning.

The dynamic behaviour of a system is expressed in TLA^+ as a transition system, with an initial state predicate, and *actions* to describe the transitions. An action formula describes the changes of state variables after a transition. In an action formula, x denotes the value of a variable x in the origin state, and x' denotes its value in the destination state. A prime is never used to distinguish symbols but always means “in the next state”. $\text{ENABLED } A$ is a predicate which is true in a state iff the action A is feasible, i.e. there exists a next state such that A is true.

A specification of a system is written as $Init \wedge \Box[Next]_{vars} \wedge \mathcal{F}$, where $Init$ is a predicate specifying the initial states, \Box is the temporal operator which asserts that the formula following it is always true, $Next$ is the transition relation, usually expressed as a disjunction of actions, $[Next]_{vars}$ is defined to equal $Next \vee vars' = vars$ ($Next$ with stuttering), and \mathcal{F} expresses fairness conditions. Fairness is usually expressed as a conjunction of weak or strong fairness on actions $\text{WF}_{vars}(A_1) \wedge \text{WF}_{vars}(A_2) \dots \wedge \text{SF}_{vars}(A_i) \dots$. Weak fairness $\text{WF}_v(A)$ means that either infinitely many A steps occur or A is infinitely often disabled. In other words, an A step must eventually occur if A is continuously enabled. Strong fairness $\text{SF}_v(A)$ means that either infinitely many A steps occur or A is eventually disabled forever. In other words, an A step must eventually occur if A is repeatedly enabled.

System properties are specified using linear temporal logic (LTL). $\Box\phi$ means that ϕ holds in every suffix of the behaviour. $\Diamond\phi$ is defined to equal $\neg\Box\neg\phi$ and means that ϕ eventually holds in a subsequent state. $\psi \leadsto \phi$ is defined to equal to $\Box(\psi \Rightarrow \Diamond\phi)$ and means that, whenever ψ holds, then later ϕ holds.

5.2 Organisation and Structure of the TLA^+ Modules

Communication models are self-contained TLA^+ modules that respect a common pattern with:

- Two state variables *net* and *H*: the network of messages in transit and the history (local, causal, or global depending on the model)
- Two actions *send* and *receive* parameterised by the peer on which the said action happens, the concerned channel, and the listened channels (for *receive*)

This corresponds to the common layout of the seven communication models presented in Definition 24 (page 71). The TLA^+ specifications of those seven communication models are direct translations of the definition.

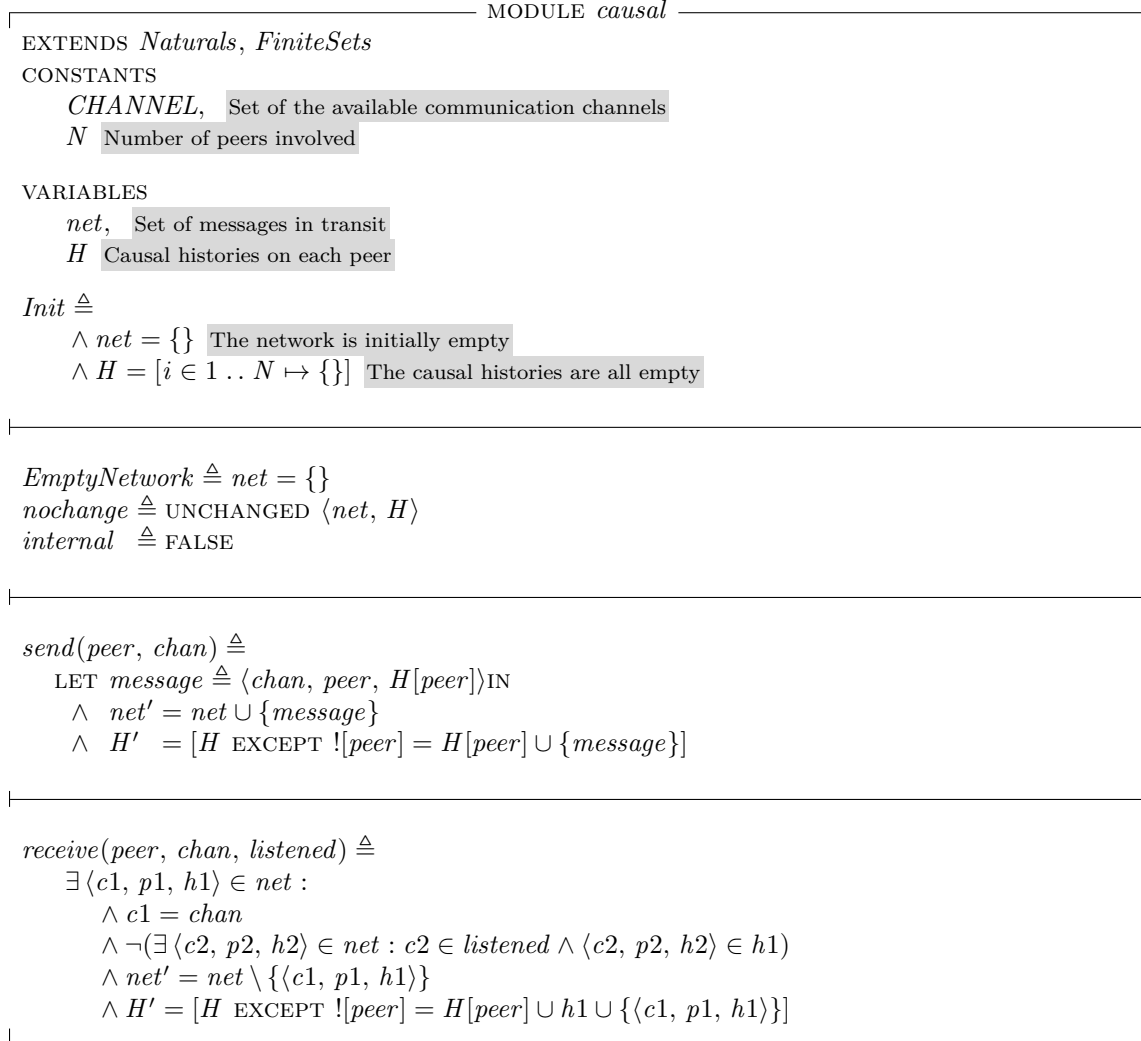


Figure 5.1: TLA⁺ Module Associated to the Causal Communication Model

As an example, Figure 5.1 shows the TLA^+ module corresponding to the causal communication model. All the models are available at:

<http://hurault.perso.enseeiht.fr/asynchronousCommunication/>.

Table 5.1 sums up the content of the TLA^+ module of each one of the seven communication models. One may of course come up with additional communication models that respect the layout. The mechanisation is not limited to the seven studied communication models.

The TLA^+ specifications of compositions of peers instantiates the TLA^+ specification of a communication model in order to refer to the predicates for the *Init*, *send* and *receive* actions of that model. Although, once again, the content of the module that specifies the composition can be written from scratch as long as it interfaces with the actions of the communication model, we provide a standard layout to describe compositions of peers specified with transition systems according to Definition 18 (page 61). This layout relies on ordinal counters to characterise the state of each peer in the corresponding transition system. An action in the composition correspond to the conjunction of a communication action and a change of the value of a peer's counter. This corresponds to the synchronisation between the transitions of the peers and the transitions of the communication model.

Figure 5.2 shows an example of the TLA^+ module of a simple composition under the *Causal* communication model. It extends a TLA^+ module called *peermanagement*, presented in Figure 5.3, that eases the management of the counters that characterise the state of the peers in a composition. In particular it provides the *trans* action to change the state of a peer and defines temporal properties that serve as a basis for compatibility checking. Most compatibility criteria described in Definition 23 (page 23) are covered by the module. Unlike the other compatibility criteria, we actually use the native detection of deadlock of TLC which checks $\text{ENABLED}(\text{Next})$. Note that *Next* may include user stuttering which is distinguished by TLC from the implicit stuttering of $[\text{Next}]_{\text{vars}}$. As for the rest of the mechanised framework, the list of compatibility properties can be extended or modified at will.

Figure 5.4 recaps the overall structure of the mechanisation with the different modules and steps of the verification. It also includes the optional user-friendly automations that the next section describes.

5.3 User-Friendly Automations

The specification with transition systems of even quite simple peers can be cumbersome. One may want to step back and provide more abstract specifications. Furthermore, in order to check the compatibility of compositions of peers, information about terminal and faulty states has to be provided. For these reasons the proposed mechanisation provides alternate ways to specify peers. The following describes how specifications of peers can be derived from CCS terms using the standard CCS rules at first, and then completed into specifications that are ready for compatibility checking. These two additional and optional steps are summed up in Figure 5.4.

5.3.1 Alternate Specification of a Peer using a CCS Term

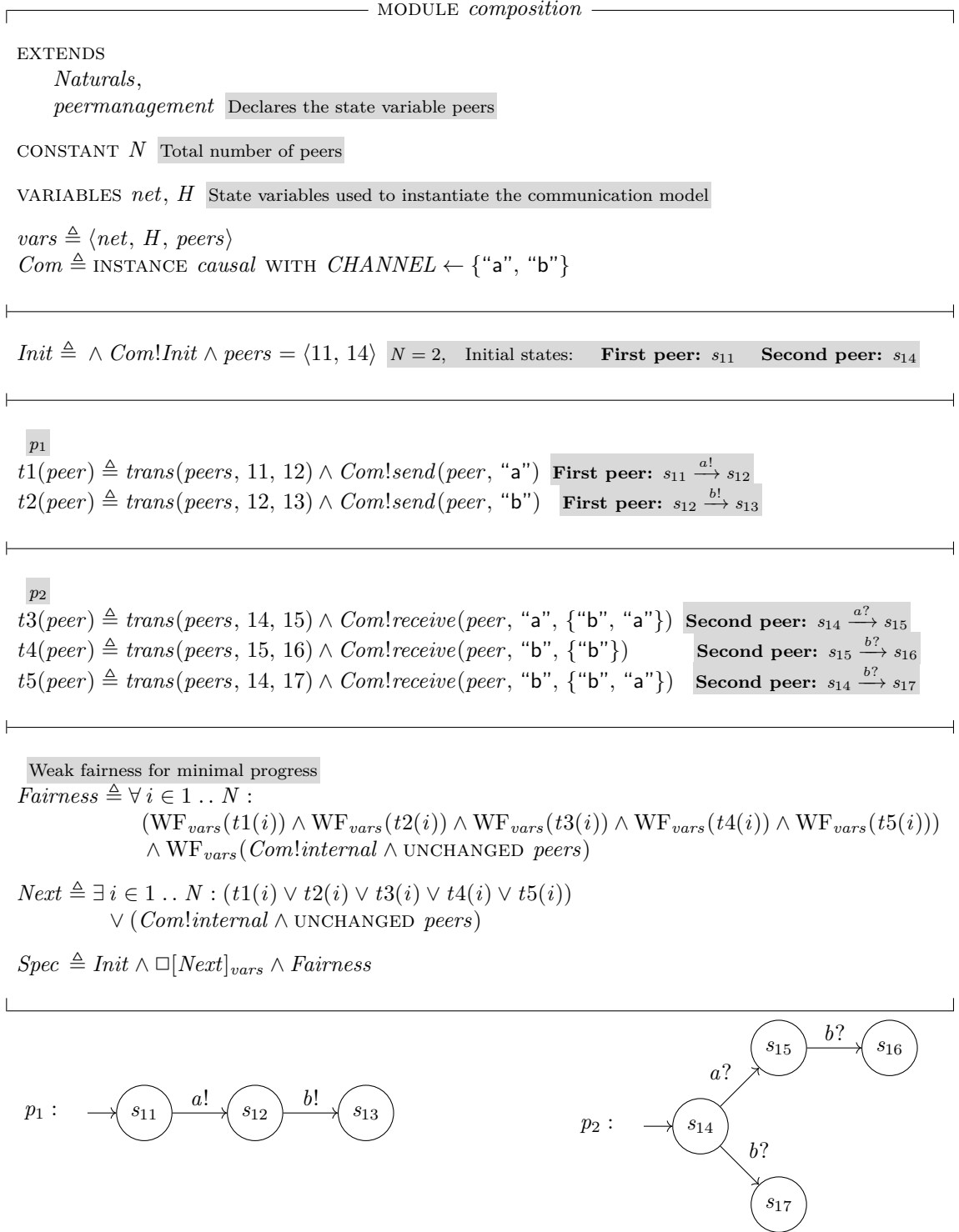
A peer can alternatively be described by a process specified with a CCS term where we consider:

- the empty process 0, neutral element of $+$ and \parallel ,
- the prefixing operator \cdot , to perform an action followed by a process. An action is τ (an internal action), or $c!$ (a send action over a channel c), or $c?$ (a receive action on c),
- the choice operator $+$,

Model	Variables	Send $s_{cm} \xrightarrow{p, c!} s'_{cm}$	Receive $s_{cm} \xrightarrow{p, c?, L} s'_{cm}$
M_{RSC}	$net, \langle H_p \rangle$	$net = \emptyset$ $\wedge net' = \{\langle c, p, H_p \rangle\}$ $\wedge H'_p = H_p \cup \{\langle c, p, H_p \rangle\}$ $\wedge \forall k \neq p : H'_k = H_k$	$\exists \langle c_1, j, h_1 \rangle \in net :$ $c_1 = c$ $\wedge net' = net \setminus \{\langle c_1, j, h_1 \rangle\}$ $\wedge \forall k : H'_k = H_k$
M_{nn}	net, H	$H' = H \cup \{\langle c, p, H \rangle\}$ $\wedge net' = net \cup \{\langle c, p, H \rangle\}$	$\exists \langle c_1, j, h_1 \rangle \in net :$ $c_1 = c$ $\wedge \neg \exists \langle c_2, l, h_2 \rangle \in net : \langle c_2, l, h_2 \rangle \in h_1$ $\wedge net' = net \setminus \{\langle c_1, j, h_1 \rangle\}$ $\wedge H' = H$
M_{1n}	$net, \langle H_p \rangle$	$H'_p = H_p \cup \{\langle c, p, H_p \rangle\}$ $\wedge net' = net \cup \{\langle c, p, H_p \rangle\}$ $\wedge \forall k \neq p : H'_k = H_k$	$\exists \langle c_1, j, h_1 \rangle \in net :$ $c_1 = c$ $\wedge \neg \exists \langle c_2, l, h_2 \rangle \in net :$ $\wedge l = j$ $\wedge \langle c_2, l, h_2 \rangle \in h_1$ $\wedge net' = net \setminus \{\langle c_1, j, h_1 \rangle\}$ $\wedge \forall k : H'_k = H_k$
M_{n1}	net, H	$H' = H \cup \{\langle c, p, H \rangle\}$ $\wedge net' = net \cup \{\langle c, p, H \rangle\}$	$\exists \langle c_1, j, h_1 \rangle \in net :$ $c_1 = c$ $\wedge \neg \exists \langle c_2, l, h_2 \rangle \in net :$ $\wedge c_2 \in L$ $\wedge \langle c_2, l, h_2 \rangle \in h_1$ $\wedge net' = net \setminus \{\langle c_1, j, h_1 \rangle\}$ $\wedge H' = H$
M_c	$net, \langle H_p \rangle$	$H'_p = H_p \cup \{\langle c, p, H_p \rangle\}$ $\wedge net' = net \cup \{\langle c, p, H_p \rangle\}$ $\wedge \forall k \neq p : H'_k = H_k$	$\exists \langle c_1, j, h_1 \rangle \in net :$ $c_1 = c$ $\wedge \neg \exists \langle c_2, l, h_2 \rangle \in net :$ $\wedge c_2 \in L$ $\wedge \langle c_2, l, h_2 \rangle \in h_1$ $\wedge net' = net \setminus \{\langle c_1, j, h_1 \rangle\}$ $\wedge H'_p = H_p \cup h_1 \cup \{\langle c_1, j, h_1 \rangle\}$ $\wedge \forall k \neq p : H'_k = H_k$
M_{11}	$net, \langle H_p \rangle$	$H'_p = H_p \cup \{\langle c, p, H_p \rangle\}$ $\wedge net' = net \cup \{\langle c, p, H_p \rangle\}$ $\wedge \forall k \neq p : H'_k = H_k$	$\exists \langle c_1, j, h_1 \rangle \in net :$ $c_1 = c$ $\wedge \neg \exists \langle c_2, l, h_2 \rangle \in net :$ $\wedge l = j$ $\wedge c_2 \in L$ $\wedge \langle c_2, l, h_2 \rangle \in h_1$ $\wedge net' = net \setminus \{\langle c_1, j, h_1 \rangle\}$ $\wedge \forall k : H'_k = H_k$
M_a	$net, \langle H_p \rangle$	$H'_p = H_p \cup \{\langle c, p, H_p \rangle\}$ $\wedge net' = net \cup \{\langle c, p, H_p \rangle\}$ $\wedge \forall k \neq p : H'_k = H_k$	$\exists \langle c_1, j, h_1 \rangle \in net :$ $c_1 = c$ $\wedge net' = net \setminus \{\langle c_1, j, h_1 \rangle\}$ $\wedge \forall k : H'_k = H_k$

(Initially, all vars are equal to \emptyset .)

Table 5.1: Specification of the Actions in the TLA⁺ modules of the Communication Models based on Definition 24 (page 71).



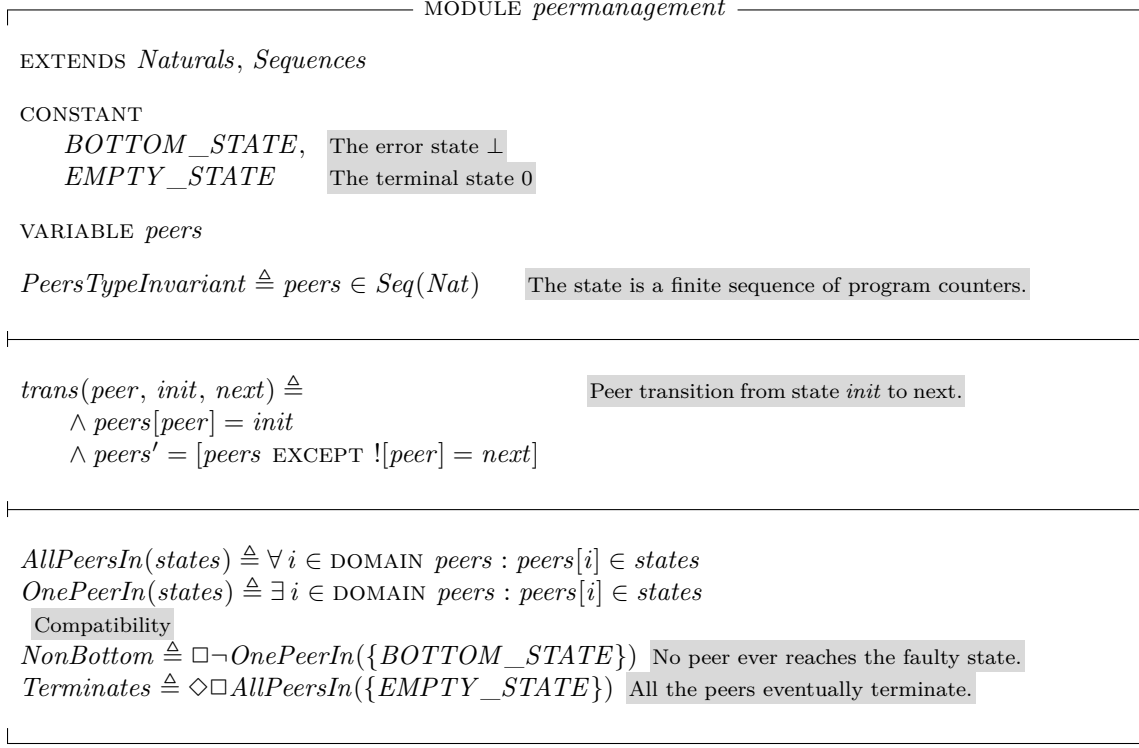


Figure 5.3: TLA⁺ Module that Ease the Management of the Peers in a Composition. It provides the *trans* action to change the state of a peer and defines temporal properties that might serve as a basis for compatibility checking.

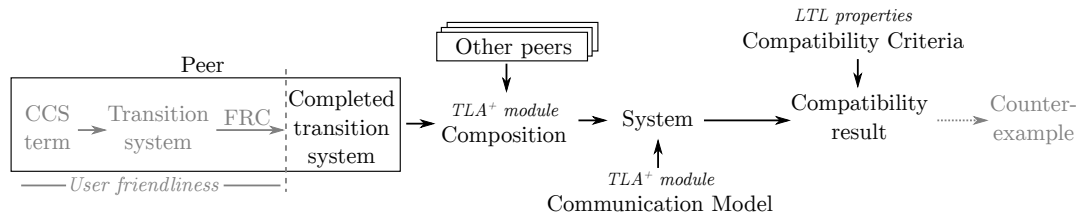
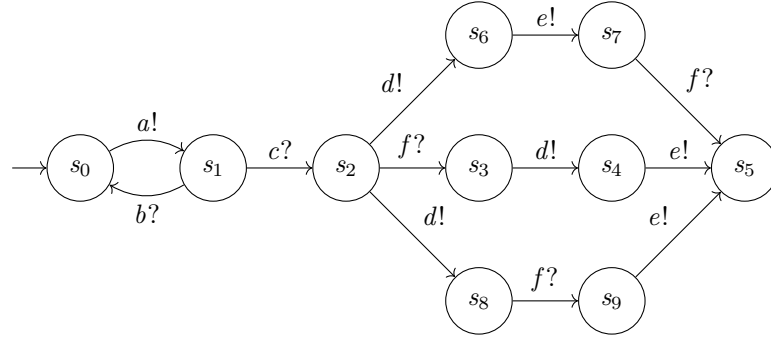


Figure 5.4: Main Steps Performed by the Framework Including Optional User Friendly Steps



$$X \triangleq a! \cdot \left(b? \cdot X + (c? \cdot (d! \cdot e! \cdot 0 \parallel f? \cdot 0)) \right)$$

Figure 5.5: Example of a Specification of a Peer Derived from a CCS Term

- the parallel composition operator \parallel ,
- the restriction operator \backslash ,
- the relabelling operator $[\]$,
- and process identifiers (defined by $X \triangleq Process$).

The peer transition system is derived from the CCS term using the standard CCS rules [Mil99, p.39], excluding the synchronous communication rule $REACT_t$. Since synchronous communication is not used, \parallel is similar to an interleaving operator. It can model internal parallelism and dynamic creation of processes inside a peer. The translation from a CCS term to a transition system is achieved through the Edinburgh Concurrency Workbench [CPS93]. We directly use the ability of The Concurrency Workbench to output the transition system of a CCS term (command **graph** without any change to the software). Each peer is independently translated, and as the translation is not applied to the composed system (the set of peers), synchronous communication (*reaction* in Milner’s vocabulary) does not occur, and only *observable actions* (Milner’s vocabulary) appear in the translation.

Figure 5.5 provides an example of a specification of a peer derived from the CCS term $X \triangleq a! \cdot \left(b? \cdot X + (c? \cdot (d! \cdot e! \cdot 0 \parallel f? \cdot 0)) \right)$.

On-the-fly construction of the transition systems would make incompatibility detection more efficient as the complete transition system may be unnecessary for a counter-example, but proving compatibility would still require constructing all transitions of the peers. PlusCal specifications [Lam09], for instance, could also offer practical alternatives to CCS and the explicit generation of transitions.

5.3.2 Faulty Reception Completion

Specifying a peer that is stable with regard to interest according to Definition 20 (page 63) requires to take all the possible premature receptions into account, in every state of the peer. This can be cumbersome and prone to error. Yet, as detailed in the previous chapter, this property is crucial to ensure that the specifications of the communication models conform to the ordering policies they are supposed to guarantee. The Faulty Reception Completion (FRC)

consists in identifying the unexpected receptions in a peer and mark them as faulty. This means it adds the corresponding transitions toward the faulty state \perp .

Informally, for every state s that is the origin of a $(\text{Receive}, _)$ transition, the set of channels corresponding to possible future receptions is computed (the future channels). For each one of these channels c , unless there is already a corresponding $(s, (\text{Receive}, c), _)$ transition, a new transition $(s, (\text{Receive}, c), \perp)$ is added to the specification. We call such transitions “faulty receptions”. Eventually, the obtained specification is stable with regard to interest: once a peer is not interested in a channel, it shall never be again later.

Definition 37 (Faulty Reception Completion). *Let $P = (S, I, R, L)$ be the specification of a peer.*

$$\text{FRC}(P) \triangleq (S \cup \{\perp\}, I, R_{\text{FRC}}, L)$$

with:

$$R_{\text{FRC}} \triangleq R \cup \left\{ (s, (\text{Receive}, c), \perp) \mid \left(\begin{array}{l} s \in \text{ReceiveStates}_P(S) \\ \wedge \quad c \in \text{FutureChannels}_P(s) \setminus \text{ListenedChannels}_P(s) \end{array} \right) \right\}$$

where:

- $\text{ReceiveStates}_P(S) \triangleq \{s \in S \mid \exists c \in C, s' \in S : s \xrightarrow{c?} s' \in R\}$ are the states in S having at least one receive transition.
- $\text{FutureChannels}_P(s) \triangleq \{c \in C \mid \exists s_1, s_2 \in S : s \rightarrow s_1 \in R^* \wedge s_1 \xrightarrow{c?} s_2 \in R\}$ (with R^* the reflexive transitive closure of $\{s \rightarrow s' \in S^2 \mid \exists l \in L : s \xrightarrow{l} s' \in R\}$). are the channels of future possible receptions from s .
- $\text{ListenedChannels}_P(s) \triangleq \text{LC}_P(s) \triangleq \{c \in \mathcal{C} \mid \exists s' \in S : s \xrightarrow{c?} s' \in R\}$ are the listened channels in state s according to Definition 19 page 63:

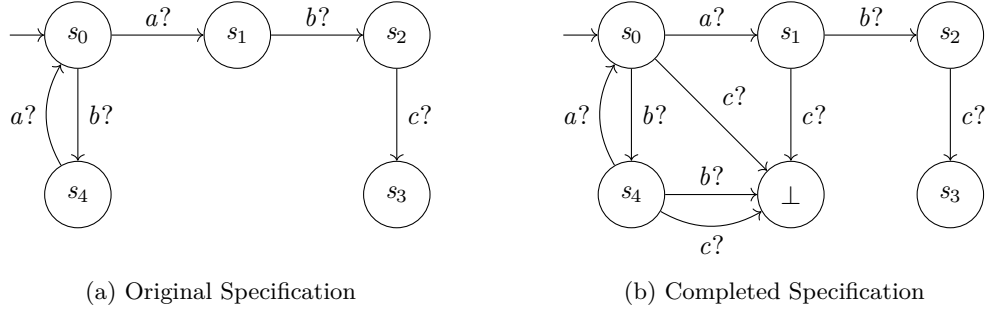
For instance, let us consider the specification of the peer depicted in Figure 5.6a. In state s_4 , the future channels are a , b , and c because a reception from a is possible in state s_4 which leads to state s_0 . From state s_0 , a reception from b is possible and s_2 is a reachable state. In s_2 , a reception from c is possible. However, the only specified reception in s_4 is from channel a . This means state s_4 misses two reception transitions: one from channel b , the other from channel c . In the completed specification in Figure 5.6b, two corresponding faulty transitions have been added. In the example, the other missing faulty receptions were from channel c on states s_0 and s_1 .

Theorem 38 (Faulty Reception Completion provides Stability with regard to Interest). *Given P the specification of a peer, $\text{FRC}(P)$ is stable with regard to interest.*

Proof. Let (S, I, R, L) the specification of $\text{FRC}(P)$. By definition of $\text{FutureChannels}_{\text{FRC}(P)}$, $\forall s, s' \in S : (s, s') \in R^* \Rightarrow \text{FutureChannels}_{\text{FRC}(P)}(s') \subseteq \text{FutureChannels}_{\text{FRC}(P)}(s)$, and by construction of $\text{FRC}(P)$, $\forall s \in S : \text{LC}_{\text{FRC}(P)}(s) = \text{FutureChannels}_{\text{FRC}(P)}(s)$. \square

5.3.3 Composite Communication Models

Up to this point systems are composed of a set of peers associated to a communication model that ensures ordering properties on the communication medium. Messages transiting between the



State	Future Channels	Listened Channels	Missing Receptions
s_0	$\{a, b, c\}$	$\{a, b\}$	$\{c\}$
s_1	$\{b, c\}$	$\{b\}$	$\{c\}$
s_2	$\{c\}$	$\{c\}$	$\{\}$
s_3	$\{\}$	$\{\}$	$\{\}$
s_4	$\{a, b, c\}$	$\{a\}$	$\{b, c\}$

Figure 5.6: Example of Faulty Reception Completion

peers on all the channels involved in the communication are handled by a unique communication model. However, some practical cases require that different sets of channels be associated to different instances of communication models. This motivates the specification of composite communication models.

Messages on channels that are associated to only one model are emitted on (resp received from) one instance of that model. Messages on channels that are associated to several models are simultaneously emitted on (resp received from) instances of these models. Therefore, the reception of a message on such a channel can only occur when the ordering properties of all the involved communication models are met. For instance, if a , b , and c are channels associated to an instance of *Causal* and c , d to an instance of *FIFO 1-1*, then the reception of a message from c will require that it respects the causality of the emissions on a , b , and c , while also respecting the order of emissions on c and d from the same peer.

When a message is received from a channel, it has to be retrieved from every communication model instance it is associated to. In each of these instances, the same message can be locally identified by a different history. To prevent inconsistent receptions, messages are given a global id shared in the different instances.

Figure 5.7 shows the generated TLA^+ module associated to a composite communication model that takes into account channels that can be associated to the *FIFO 1-1* communication model (channels exclusively in *CH1*), the causal communication model (channels exclusively in *CH2*), or both (channels in *CH1* and *CH2*). The referenced primitive communication models TLA^+ specifications have to handle message identifiers in the communication actions. Since they are external markers used by composite communication models, it does not change the structure of the derived transition system nor the ordering properties it guarantees.

MODULE *multicom*

EXTENDS *Naturals*
CONSTANTS N ,
 $CH1$, Channels of the first communication model
 $CH2$ Channels of the second communication model
VARIABLES $messid, net1, H1, net2, H2$
 $Var1 \triangleq \langle net1, H1 \rangle$ State variables of the first communication model
 $Var2 \triangleq \langle net2, H2 \rangle$ State variables of the second communication model
 $Vars \triangleq \langle Var1, Var2, messid \rangle$
 $Com1 \triangleq$ INSTANCE *fifo11* WITH $CHANNEL \leftarrow CH1, net \leftarrow net1, H \leftarrow H1$
 $Com2 \triangleq$ INSTANCE *causal* WITH $CHANNEL \leftarrow CH2, net \leftarrow net2, H \leftarrow H2$

$Init \triangleq messid = [chan \in CH1 \cup CH2 \mapsto \{\}] \wedge Com1!Init \wedge Com2!Init$

$max_id(chan) \triangleq$ CHOOSE $n \in messid[chan] : \forall p \in messid[chan] : p \leq n$
 $available_id(chan) \triangleq$ IF $messid[chan] = \{\}$ THEN 1 ELSE (Message *id* generation and reuse
CHOOSE $n \in 1 \dots max_id(chan) + 1 :$
 $(n \notin messid[chan] \wedge (\forall p \in 1 \dots n - 1 : p \in messid[chan]))$)

$EmptyNetwork \triangleq Com1!EmptyNetwork \wedge Com2!EmptyNetwork$
 $nochange \triangleq$ UNCHANGED $messid \wedge Com1!nochange \wedge Com2!nochange$
 $internal \triangleq \wedge$ UNCHANGED $messid$
 $\wedge ((Com1!internal \wedge Com2!nochange) \vee (Com2!internal \wedge Com1!nochange))$

$send(peer, chan) \triangleq$
LET $id \triangleq available_id(chan)$ IN
 $\wedge messid' = [messid \text{ EXCEPT } ![chan] = @ \cup \{id\}]$ Take available *id*
 \wedge On a channel from second group only: send on *Com2*
 $\vee (chan \notin CH1 \wedge chan \in CH2 \wedge \text{UNCHANGED } Var1 \wedge Com2!send(peer, chan, id))$
On a channel from first group only: send on *Com1*
 $\vee (chan \in CH1 \wedge chan \notin CH2 \wedge \text{UNCHANGED } Var2 \wedge Com1!send(peer, chan, id))$
On a channel from both group: send simultaneously (same message *id*) on *Com1* and *Com2*
 $\vee (chan \in CH1 \wedge chan \in CH2$
 $\wedge Com1!send(peer, chan, id) \wedge Com2!send(peer, chan, id))$

$receive(peer, chan, listened) \triangleq$
 $\exists id \in messid[chan] :$
 $\wedge messid' = [messid \text{ EXCEPT } ![chan] = @ \setminus \{id\}]$ Release message *id* for reuse
 \wedge On a channel from second group only: receive from *Com2*
 $\vee (chan \notin CH1 \wedge chan \in CH2 \wedge \text{UNCHANGED } Var1$
 $\wedge Com2!receive(peer, chan, id, listened))$
On a channel from first group only: receive from *Com1*
 $\vee (chan \in CH1 \wedge chan \notin CH2 \wedge \text{UNCHANGED } Var2$
 $\wedge Com1!receive(peer, chan, id, listened))$
On a channel from both group: receive simultaneously (same message *id*) from *Com1* and *Com2*
 $\vee (chan \in CH1 \wedge chan \in CH2$
 $\wedge Com1!receive(peer, chan, id, listened) \wedge Com2!receive(peer, chan, id, listened))$

Figure 5.7: Composite Communication Model with *FIFO 1-1* and *Causal*

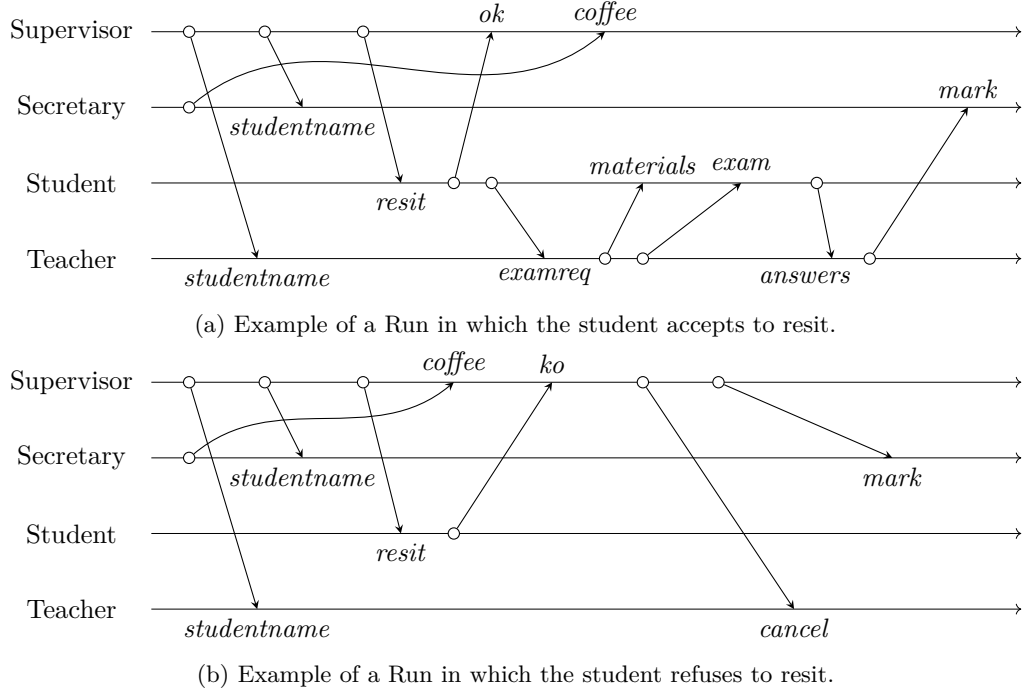


Figure 5.8: Examples of Expected Runs of the University Example

5.4 Examples and Results

5.4.1 Detailed Example: The Examination Management System

Specification

Let us consider an examination management system composed of a student, a supervisor, a secretary, and a teacher. When a student has failed, the supervisor sends their name to the teacher and the secretary, and the resit information to the student. A student who chooses to resit answers *ok* and asks the teacher for the exam. The teacher sends the needed materials and then the exam, after which the student sends back answers. Eventually, the teacher sends a mark to the secretary. A student who declines to resit informs the supervisor who sends a cancel message to the teacher and the former mark to the secretary. An unrelated exchange also occurs between the secretary and the supervisor who would like to meet during the coffee break. The secretary sends a message to inform the supervisor that coffee is ready. The supervisor is ready to join after having sent work-related messages: just before, after, or while dealing with the student's choice. Sample executions are depicted in Figure 5.8a and Figure 5.8b. The system is specified in Figure 5.9.

Next, consider the properties needed to make this work as intended. There is a causal dependency between the *studentname* message and the *examreq* message (the request for the exam must not arrive before the student name). This causal dependency comes from the *resit* message, which follows the *studentname* message and is the cause of the *examreq* message. Causal communication is thus required. Moreover, if a *cancel* message is sent, it should be received after the student's name by the teacher. Therefore, *cancel* is part of this causal group. The same holds

$$\begin{aligned}
\text{Supervisor} &\triangleq \text{studentname!} \cdot \text{studentname!} \cdot \text{resit!} \cdot \text{SupervisorWaiting} \\
\text{SupervisorWaiting} &\triangleq \left(\begin{array}{c} \left(\begin{array}{c} \text{ok?} \cdot 0 \\ + \text{ko?} \cdot \text{cancel!} \cdot \text{mark!} \cdot 0 \end{array} \right) \\ \parallel (\text{coffee?} \cdot 0) \end{array} \right) \\
\text{Secretary} &\triangleq \text{coffee!} \cdot \text{studentname?} \cdot \text{mark?} \cdot 0 \\
\text{Student} &\triangleq \text{resit?} \cdot \left(\begin{array}{c} \tau \cdot \text{ko!} \cdot 0 \\ + \tau \cdot \text{ok!} \cdot \text{StudentOk} \end{array} \right) \\
\text{StudentOk} &\triangleq \text{examreq!} \cdot \text{materials?} \cdot \text{exam?} \cdot \text{answers!} \cdot 0 \\
\text{Teacher} &\triangleq \text{studentname?} \cdot \left(\begin{array}{c} \text{cancel?} \cdot 0 \\ + \text{examreq?} \cdot \text{TeacherExam} \end{array} \right) \\
\text{TeacherExam} &\triangleq \text{materials!} \cdot \text{exam!} \cdot \text{answers?} \cdot \text{mark!} \cdot 0
\end{aligned}$$

Figure 5.9: Supervisor-Secretary-Student-Teacher Specification

for the *mark* channel, since the secretary first expects a *studentname*. Besides, the *materials* and the *exam* are sent in two separate messages and are expected to be received in this order by the student. Lastly, the coffee break exchange requires that several messages can be in transit so that the supervisor can send the *studentname* and *resit* messages after the secretary has sent *coffee*.

This example is checked with the seven asynchronous models of Table 5.1, and with the following composite model:

<i>Causal</i>	M_c	$\{\text{studentname}, \text{resit}, \text{examreq}, \text{cancel}, \text{mark}\}$
<i>FIFO 1-1</i>	M_{11}	$\{\text{materials}, \text{exam}\}$
<i>Fully Asynchronous</i>	M_a	$\{\text{ok}, \text{ko}, \text{answers}, \text{coffee}\}$

Compatibility

In this example, *studentname* is a channel over which two messages are sent and from which they are received by different services (teacher and secretary). In addition, *mark* is a channel over which only one message is to transit, but it may be emitted by different services (supervisor and teacher). Therefore, compatibility, especially termination of the secretary service, is not trivial. Consequently, in addition to the generic compatibility properties defined in Definition 23 (page 68), we also consider the termination of the secretary and we check if all messages have been received upon full termination.

Figure 5.10 presents the results. It confirms that *Causal* (or a model stricter than *Causal* apart from *RSC*) is required to ensure compatibility of the composition. However, causality is not required over the whole set of channels. The composite model with the considered partition is a restrictive enough communication model. In this example, the maximal number of distinct states is 988.

	M_{RSC}	M_{nn}	M_{1n}	M_{n1}	M_c	M_{11}	M_a	$M_{composite}$
Termination	×	✓	✓	✓	✓	×	×	✓
Termination (empty network)	×	✓	✓	✓	✓	×	×	✓
Termination of the secretary	×	✓	✓	✓	✓	×	×	✓
No faulty receptions	✓	✓	✓	✓	✓	×	×	✓
No communication deadlock	×	✓	✓	✓	✓	✓	✓	✓

Figure 5.10: Compatibility Results: Examination Management System

	M_{RSC}	M_{nn}	M_{1n}	M_{n1}	M_c	M_{11}	M_a	$M_{composite}$
No faulty receptions	✓	✓	✓	✓	✓	×	×	✓
Messages will be received								
$\forall m \in \mathcal{M} :$	✓	✓	✓	✓	✓	×	×	✓
$m \in net \leadsto m \notin net$								

Figure 5.11: Compatibility Results: Client-Controller-Application System

5.4.2 Practical Example: The Client-Controller-Application System

Let us consider an example taken from [SHQ17] where a client interacts with an application and a third peer, the controller, to get the authorisation to access the application. The controller starts the application when needed. More precisely, the client sends a *login* to the controller which can *accept* or *reject* the demand. If accepted, the client can send several *upload* messages to the application. This controller starts the application (message *begin*) when it accepts a client, and signals it to *end* when the client logs out (*logout*). Figures 5.12a, 5.12b, and 5.12c specify the peers. They are stable with regard to interest.

In terms of unexpected receptions, the framework highlights counterexamples. When the communication is not *Causal* or stronger (according to the hierarchy in Section 2.5 of Chapter 2 on page 38) on channels *begin*, *accept*, and *upload*, a message sent on *begin* by the controller might not be received by the application (in state ap_1) before the message sent by the client on *upload*. This leads the application in the faulty state \perp . Similarly, when the communication is not *FIFO 1-1* or stronger on *login* and *logout*, the client can log in just after logging out but if the message on *login* is received first, the application remains in ap_2 and the controller can send a message on *begin* that leads the application in the faulty state \perp upon reception. Figure 5.11 recaps these results. The composite communication model is:

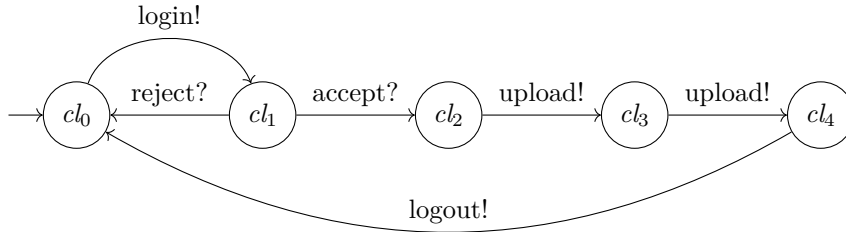
<i>Causal</i>	M_c	$\{begin, accept, upload\}$
<i>FIFO 1-1</i>	M_{11}	$\{login, logout\}$
<i>Fully Asynchronous</i>	M_a	$\{reject\}$

5.4.3 Advanced Usage of Composite Models: the Video Stream

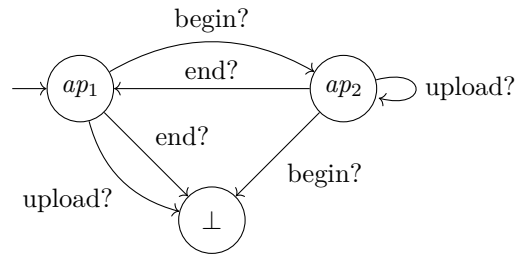
The previous examples has illustrated how channels can be partitioned and associated to different communication models in order to perform sharper analysis. Here we provide a case where the need for channels to be associated to more than one communication model arises.

Let us consider a system in which a client watches a live video with subtitles. The video is stored on a remote video server, and captioning is performed on the fly by a subtitle generator. The streams are cut into video parts (resp. subtitle parts) denoted V_i (resp. S_i) where i designates the i -th part. The client expects to receive each video part, one after the other, and the associated subtitle part with little enough delay between them.

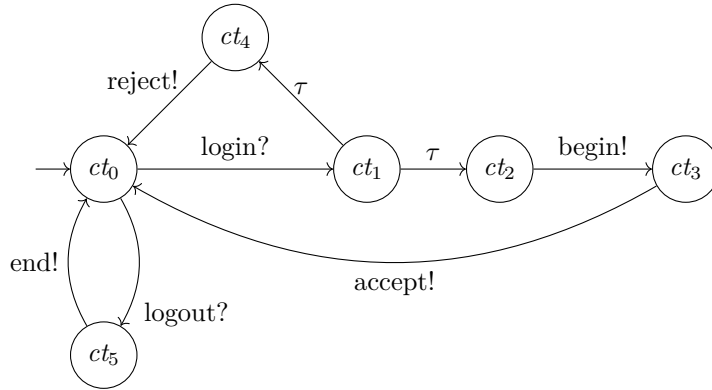
In order to achieve that goal, we introduce checkpoints in the streams. Each video or subtitle



(a) Client



(b) Application



(c) Controller

Figure 5.12: Specifications of the Client, Controller, and Application. The peers are stable with regard to interest.

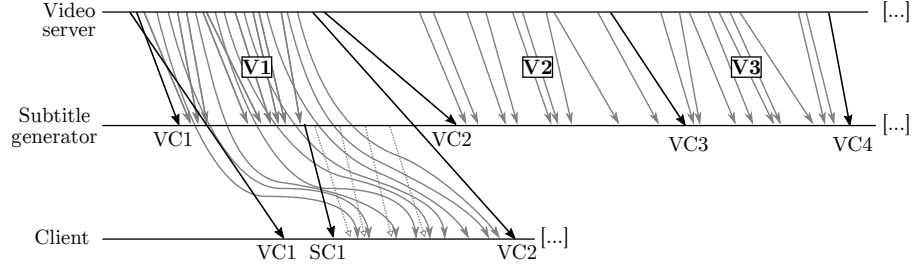


Figure 5.13: Transmission of One Part Between the Three Peers

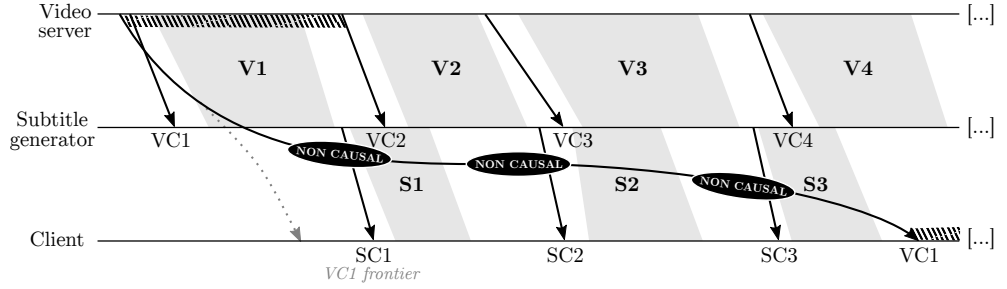


Figure 5.14: Desirable and Undesirable Executions with FIFO 1-1

part V_i (resp. S_i) is preceded by the emission of an associated checkpoint message denoted VC_i (resp. SC_i). *FIFO 1-1* ordering is used so that messages that compose a part are received in the right order, as well as the checkpoint messages. Figure 5.13 provides a possible execution where a part is transmitted as expected.

FIFO 1-1 communication is not sufficient to prevent the video stream from being late with regard to the subtitles, as they are sent by different peers (Figure 5.14). Causal communication on the entire system would prevent this, but this is excessive, as a little desynchronisation between a video part and its corresponding subtitle part is not perceived. It is sufficient to consider a composite communication model, where all the channels are associated to an M_{1-1} instance, and all the control point channels are also associated to an M_{causal} instance. Model checking of this system (three parts of two V_i/S_i messages each, plus checkpoint messages, for a total of 30 messages) generates 353988 distinct states and takes around 22s with an optimised implementation of M_{1-1} .

5.5 Optimised Communication Models

Several practical issues and restrictions arise from the provided specifications of the communication models. For instance, as the presented communication models are based on message histories that never decrease, we are confronted to performance limitations. Moreover, systems involving behaviours with loops, such as for instance the trivial system composed of any communication model and two peers derived from the CCS terms $P_1 \triangleq a! \cdot b? \cdot P_1$ and $P_2 \triangleq a? \cdot b! \cdot P_2$, will actually consist in an infinite transition system.

5.5.1 Reduction to Finite State Spaces by Purging Histories

One solution to the problem of systems with loops could consist in using unique message identifiers as a base for message histories. When a message is sent, it is given a new unique identifier and this very identifier is added to the history (local, global, or causal) instead of the message itself. This means that the history associated to a message in transit is a simple flat set of identifiers instead of a recursive data structure that contains entire messages. When remembering a message becomes irrelevant (no message in transit refers to it), the associated identifier can be used again for new messages. In some cases, this might change an infinite system into a finite state system and thus allow model checking. There are however two caveats:

- Such specifications introduce a new concept: message identifiers. They also change the fundamental data structure on which the communication models rely: message histories. This is a significant step away from the formal specifications from Chapter 4 that have been proven correct and complete with regard to the structural definitions from Chapter 2.
- The benefits in term of state space reduction strongly depend on the way the message identifiers are generated and reused. There are many possible ways to reuse identifiers: in a LIFO manner (the new message takes the newest available identifier), in a FIFO manner (the new message takes the oldest available identifier), only once a given amount of identifiers has been made available (the new message takes a new identifier unless there are too many available "second-hand" identifiers), etc. Some policies might increase the number of state even more: at worst a new message can take any of the available identifiers. Depending on the considered system, some choices might provide better results than others.

A mechanism to purge history has been implemented instead. In the modified implementations, messages that are retrieved from the network of a communication model are recursively removed from histories. This purge reduces the previous example to a finite state system over which model checking always reaches a conclusion. These alternative communication model specifications have proven useful to check practical examples.

5.5.2 Dedicated Optimised Implementations

When possible, more optimised and practical implementations consist in using counters of messages instead of message histories, or explicit sequences (in *FIFO* communication models). However, unlike the already presented specifications with message histories, these implementations are not guaranteed to be equivalent to the structural definitions from Chapter 2. Figure 5.15 is a TLA⁺ module that corresponds to a *FIFO* n - n specification relying on a unique queue of messages. There are not histories anymore and the network is the queue. In Chapter 6, we introduce more examples and prove they do not violate the ordering policies.

5.6 Benchmarking

5.6.1 Scenario

We consider two parameters $n \geq 0$ and $m \geq 0$, a_1, \dots, a_n and b channels. Two peers are specified by the following CCS terms from which we derive transition systems and apply FRC:

$$(a_1! \cdot \dots \cdot a_n! \cdot b? \cdot 0)^m \text{ and } (a_1? \cdot \dots \cdot a_n? \cdot b!)^m$$

MODULE *fifonnsequence*

EXTENDS *Naturals, Sequences*

CONSTANT
 $CHANNEL, N$

VARIABLE
 net , The network is now a sequence of messages
 H There is no need for histories (still present for integration in the framework)

$Init \triangleq$
 $\wedge net = \langle \rangle$ The network is initially the empty sequence
 $\wedge H = \{\}$

$EmptyNetwork \triangleq net = \langle \rangle$
 $nochange \triangleq UNCHANGED \langle net, H \rangle$
 $internal \triangleq FALSE$

$send(peer, chan) \triangleq$
 $\wedge net' = Append(net, \langle chan, peer \rangle)$
 $\wedge UNCHANGED H$

$receive(peer, chan, listened) \triangleq$
 $\wedge net \neq \langle \rangle$
 $\wedge Head(net) = \langle chan, peer \rangle$
 $\wedge net' = Tail(net)$
 $\wedge UNCHANGED H$

Figure 5.15: TLA⁺ Module Associated to an Optimised Specification of the *FIFO n-n* Communication Model

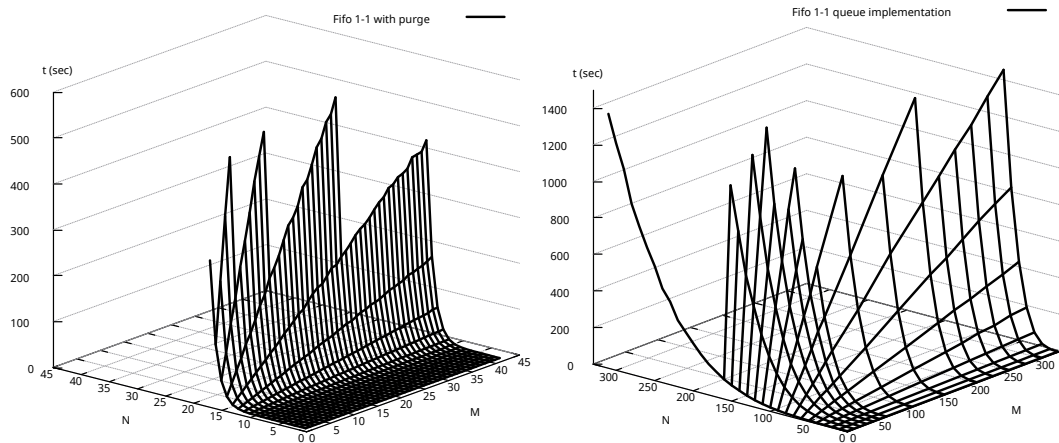


Figure 5.16: Optimised Implementations

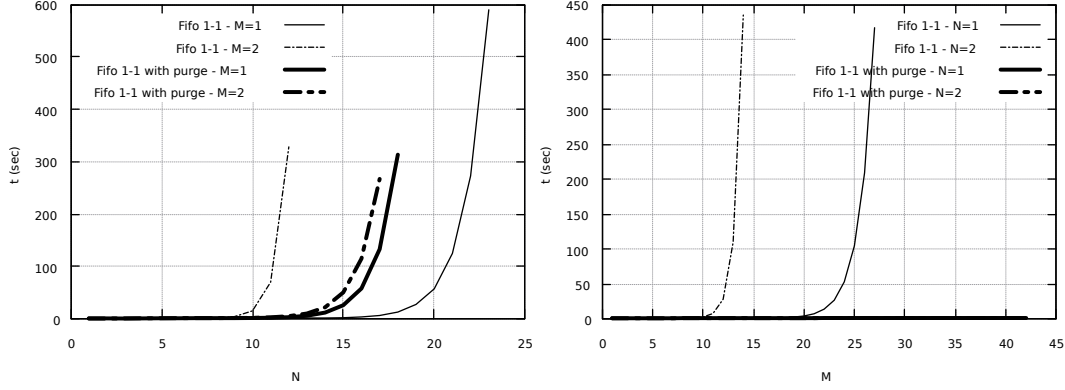


Figure 5.17: Comparison Between the Formal Implementation and the Purge-Based Optimised Implementation

M	1	1	41	51	101	301
N	1	311	111	51	91	1
states	9	48834	259494	70334	432184	1209
transitions	11	97041	509801	135361	845781	1211

Table 5.2: Number of States and Transitions (Samples)

The composition consists in transmitting sequences of n messages (on the a_i channels) from the first to the second peer. A synchronisation message is exchanged (from the second to the first peer) between each sequence. The messages are expected to be received in the order of their emission and a communication model has to ensure that unexpected receptions are impossible.

The framework is used to check the termination of the proposed composition. In the following results, we consider M_{11} , an alternative specification with history purge M_{11}^{purge} , and an implementation using a queue M_{11}^{queue} . The machine that runs the simulations is 2×4 cores Intel Xeon CPU E5-2690 v3 at 2.60GHz with 23GiB of memory. We focus on the number of generated states and the time required to perform model checking. The results are presented in Figures 5.16 and 5.17 and Table 5.2 for M_{11} , M_{11}^{purge} , and M_{11}^{queue} . The results obtained with the other models of Table 5.1 and their different implementations are similar. Indeed, their specification do not differ much in terms of data structure and guard on reception, and the time required to explore the state space do not vary significantly. Differences appear between the different implementations of a given model.

5.6.2 Analysis

With M_{11}^{purge} , the results show that the number of states (Table 5.2), which does not depend on the implementation of the model, and runtime (which is directly related to the number of states) linearly increase with m the number of critical sequences. In Figure 5.16, the runtime is plotted for different values of n and m . For a given value of n , it shows that the runtime increases linearly along the m axis. The number of states and runtime exponentially increase when it comes to n as illustrated once again on Figure 5.16. It accounts for the maximum number of messages in transit at a given time and all the possible receptions that have to be visited. m corresponds to the number of repetitions of the scenario, thus the linear profile. The second graph in Figure 5.17 reveals that runtime does not grow linearly for m with M_{11} : it grows ex-

ponentially because histories of past iterations actually accumulate which generates new states. When the histories are purged, the number of states is capped: in the same graph we see that, however large m is, runtime remains constant and looks close to 0 (almost along the m axis in the figure). Checking if a reception is possible requires to explore this entire past whereas M_{11}^{purge} and M_{11}^{queue} do not keep track of received messages.

	Advantages	Drawbacks
Regular specifications with message histories	Direct translation of the formal specification.	Keeps track of every emitted message: poor m -type scalability.
Regular specifications with purge of histories	Close to the formal specification. No accumulation of messages in histories. Better m -type scalability (linear).	Additional time needed to purge histories: state space explosion occurs earlier in terms of maximum number of messages in transit.
Dedicated specifications	Way better overall performances. Useful as a quick first compatibility check or to find potential counterexamples.	Equivalence to the formal specification is not guaranteed.

5.7 Conclusion

The mechanisation of the framework provides a practical toolset to verify the compatibility of compositions of peers with model checking. It is modular, extensible, and flexible: the specifications of the peers, the communication models, and the compatibility properties are all independent and share common interfaces to integrate in the mechanised framework. Moreover, we provide optional automations to ease the specification of complex compositions that satisfy stability with regard to interest. The TLA⁺ specifications of the communication models correspond to the exact specifications of the models from Chapter 4. The mechanised framework is trustworthy because these specifications conform to the logical definitions of the communication models in Chapter 2 when dealing with peers that are stable with regard to interest. The framework has been used to specify practical examples of compositions and reveal counterexamples (runs) that violate the compatibility properties. This allows to incrementally fine tune the communication model to fit the system's needs thanks to possibility to specify composite communication models in the mechanised framework.

In terms of performance, the impact of state space explosion on the overall time of model checking may be limited by optimised specifications of the communication models. However, there is less confidence in these alternate unproved specifications. Systems that loop such as the client-controller-application example may generate an infinite state space due to the ever increasing message history: this is solved with communication models that remove old messages from the histories at the cost of performance of the model checking.

Chapter 6

A Menagerie of Refinements

This chapter is a contribution to the study of the asynchronous communication models and how they are derived. Three approaches are considered to model the point-to-point asynchronous communication paradigms. In the first and most abstract one, communication models are specified as properties on the ordering of events in distributed executions as in Chapter 2. In the second approach, they are modelled with message histories which embed the dependencies in a similar way to the models in Chapter 4. In the last and more concrete approach, they are modelled using classical approaches such as counters, queues or vector clocks.

Refinement is used in several ways. A first chain of refinements introduces distributed events and the causality link between them. Then, a hierarchy of the seven point-to-point communication models from the previous chapters is established by refinement. This hierarchy can be used to choose the most adequate model for an application, i.e. the least constrained model which is sufficient to prove the correctness. Lastly, concrete models are derived by data refinement. The different approaches – from the most abstract to the most concrete – allow to choose several versions of the same communication model, either more amenable to proving or closer to a deployable implementation.

6.1 Introduction

A classic way to develop distributed algorithms is to start with a global goal, such as mutual exclusion or global agreement. A distributed version of the algorithm is then derived, either directly or by progressive transformation of the specification, e.g. by refinement. At the end, the most concrete versions have to ensure that variables are not shared by different sites, and that messages/signals/port interactions are used to communicate. This approach dates back to early work by Dijkstra [Dij83], Chandy-Misra with UNITY [CM88], Back and Kurki-Suonio with action systems [BK88], Lamport with TLA [Lam94] and TLA⁺ [Lam02]. It is still bustling in the correct-by-construction community and Event-B [Abr10] is a well-known framework which embodies this methodology. At one point of the development process, communication is explicitly introduced, to express the flow of information from one site to another. This communication later takes the form of message exchanges. When the development is thoroughly conducted with formal verification, the properties of the communication are shown to be sufficient for the correctness of the algorithm. However, it is often unclear what are the specific properties of this communication that are necessary to ensure the correctness of the algorithm. Especially, it may be difficult to replace one communication model with another without doing again the full proof or development.

The presented work aims at alleviating these difficulties for asynchronous point-to-point communication. Refinement is used in several ways. A first chain of refinements introduces distributed events and the causality link between them, and gives several models of asynchronous distributed executions: a model with abstract events and partial ordering, a linear extension of the partial ordering, and lastly a refined model with message-related events. Then, the seven point-to-point communication models from previous chapters are studied. They differ in the delivery order of messages (e.g., messages are always received in the order in which they were sent). The constraints on the delivery order are actually restrictions on the non-determinism of the deliveries. The ordering between events can be captured either explicitly or implicitly with message histories. This gives two chains of refinements. Yet, these two chains are closely related, yielding a ladder of refinements. Lastly, concrete models are derived by data refinement to get concrete models. On the whole, this allows to substitute one communication model with another, either having weaker / stronger properties, or having a more abstract / more concrete description. The overall picture of these refinements is presented in the conclusion (Figure 6.8).

The outline of this chapter is the following. Section 6.2 recalls basic definitions of the theory of distributed systems and their modelling in Event-B for asynchronous point-to-point communication with messages. Section 6.3 presents seven communication models and their modelling in Event-B with distributed executions and events. Section 6.4 presents the refinements to get models based on histories of messages. Section 6.5 refines one step further towards concrete models. Section 6.6 discusses several common points: proof effort, deadlock freedom, previous work in TLA^+ , localisation. Eventually, the conclusion draws perspectives after summing up this work.

6.2 Distributed Systems

An asynchronous message-passing distributed system is composed of a set of peers that exchange messages. This chapter considers point-to-point communication where message has exactly one sender and at most one receiver.

6.2.1 Distributed Executions

Let PEER be the set of peers, MESSAGE an enumerable set of messages identifiers, and $\text{COM} \triangleq \{\text{Send}, \text{Receive}\}$ the communication labels. In this chapter we stick to the definitions of distributed executions and runs from Chapter 2. In order to simplify the presentation of the results concerning the communication models, we do not consider internal actions (Internal). The total orders on peers \leq are also omitted. It is simply done with $\text{peer}(e_1) = \text{peer}(e_2) \wedge e_1 \prec_c e_2$ or $\text{peer}(e_1) = \text{peer}(e_2) \wedge e_1 <_\sigma e_2$ instead. As events occurring on the same peer are totally ordered and $<_\sigma$ contains \prec_c , both formulae are equivalent.

In the following, some ordering specialisation will forbid message loss or impose a deadlock. For instance, if a FIFO property is realised by counting sent and received messages, the loss of a message prevents the delivery of all following messages.

6.2.2 Event-B

A model in Event-B [Abr10], or machine, is an abstract state machine. It contains state variables v , invariants $I(v)$, and events¹. An event E parameterised by x has the form $\text{EVENT } E \text{ ANY } x$

¹Unfortunately, the word *event* is burdened with multiple meanings. In this chapter, an event can be an element of a distributed execution or a part of an Event-B machine. The context hopefully makes it clear which is which.

WHERE $G(v, x)$ THEN $A(v, x)$ END, where $G(v, x)$ are the guards of the event and $A(v, x)$ an action changing the values of v . In this chapter, actions are deterministic assignments of the form $v := \text{expr}$ or $v :| v' = \text{expr}$ where v is a state variable. Both forms can be represented using a before-after predicate $BA(v, x, v')$, where v' is the state of the variables after the action. When values for the parameters x satisfy the guards of E , E is said to be enabled and the state of the machine evolves according to the action A . Moreover, an event can contain witnesses (clause WITH) for removed parameters and variables of the machine it refines. **INITIALISATION** is a special event without parameters or guards that specifies the initial state of a machine. A machine can be related to one Event-B context, the machine **SEES** that context. The context specifies sets, constants, axioms and theorems (grouped in C) on these sets and constants. The invariants, guards, and actions in a machine can depend on the definitions provided by the context and proofs can rely on the axioms and theorems. The Rodin tool [ABH⁺10] generates proof obligations for the preservation of the invariants by the events. For an invariant $I(v)$, the INV proof obligation for an event is: $C \wedge I(v) \wedge G(v, x) \wedge BA(v, x, v') \Rightarrow I(v')$.

The main concept of the Event-B method is refinement. Consider a concrete machine M_c with variables w , an abstract machine M_a with variables v , the associated axioms C_c (resp C_a), and invariants $I_c(w)$ (resp $I_a(v)$). Abstract variables v are linked to concrete variables w by a gluing invariant $J(v, w)$. The refinement of an event E_a of M_a (with guard $G_a(v, x)$ and action $BA_a(v, x, v')$) in an event E_c of M_c (with guard $G_c(w, y)$ and action $BA_c(w, y, w')$) generates the proof obligations:

$$C_c \wedge C_a \wedge I_c(w) \wedge I_a(v) \wedge J(v, w) \wedge G_c(w, y) \wedge W(x, w, y) \Rightarrow G_a(v, x)$$

$$\left(\begin{array}{l} C_c \wedge C_a \wedge I_c(w) \wedge I_a(v) \wedge J(v, w) \\ \wedge G_c(w, y) \wedge W(x, w, y) \wedge BA_c(w, y, w') \end{array} \right) \Rightarrow \exists v'. (BA_a(v, x, v') \wedge J(v', w'))$$

$W(x, w, y)$ is the witness predicate which links abstract parameters x and concrete parameters y if they are different. The first rule means that the guard is strengthened (GRD proof obligation) and the second rule means that the abstract action is correctly simulated by the concrete one (SIM proof obligation). **skip** is the stuttering event which is implicitly refined by an event of a concrete machine that does not refine any other event. In our case, models also generate additional proof obligations like well-definedness of terms (WD/WWD proof obligation) and feasibility of non-deterministic action or witness (FIS/WFIS proof obligation).

In Event-B, $x_1 \mapsto x_2$ denotes a pair (x_1, x_2) . Relations are sets of pairs. $\text{dom}(r)$ and $\text{ran}(r)$ denote the domain and range of a relation r . $E \leftrightarrow F$ denotes the set of relations between E and F , $E \Leftrightarrow F$ the set of total surjective relations, and $E \rightarrow F$ total functions from E to F . The relation $r_1; r_2$ denotes the forward composition of relations r_1 and r_2 . “ \triangleleft ” is the domain restriction operator such that given a relation r and a set E , $E \triangleleft r \triangleq \{x \mapsto y \mid x \mapsto y \in r \wedge x \in E\}$. “ \trianglelefteq ” is the domain subtraction operator such that given a relation r and a set E' , $E' \trianglelefteq r \triangleq \{x \mapsto y \mid x \mapsto y \in r \wedge x \notin E'\}$. “ $\triangleleft\!\!\triangleleft$ ” is the overriding operator such that given relations r_1 and r_2 , $r_1 \triangleleft\!\!\triangleleft r_2 \triangleq r_2 \cup (\text{dom}(r_2) \triangleleft r_1)$. $\mathbb{P}(E)$ denotes the powerset of E and $\text{union}(E) \triangleq \cup_{X \in E} X$. $\text{partition}(E, E_1, \dots, E_n)$ is a predicate that states that E is partitioned in E_1, \dots, E_n .

6.2.3 From Events to Distributed Executions

The general architecture of the initial refinements is shown in Figure 6.1. The initial machines introduce events, and then establish a link between two events: a cause and a consequence (context **Event** and machines **Events** and **Communication**). Peers are then introduced, and events occur on peers (machine **DistributedExecution** and context **Peers**). This yields distributed executions, where events are causally related (partial order \prec_c). Next, a run is a linear extension

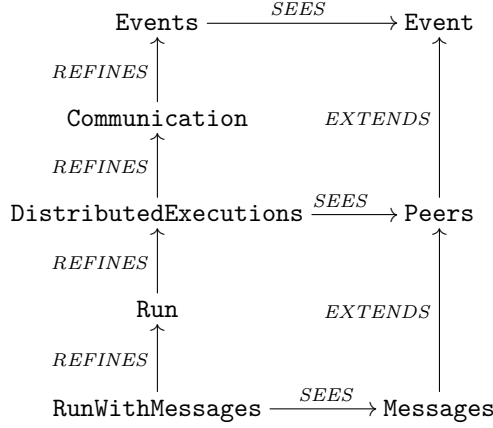


Figure 6.1: General Architecture of the Preliminary Design

of \prec_c , where events are totally ordered by $<_\sigma$ (machine **Run**). Lastly, messages are introduced (context **Messages** and machine **RunWithMessages**). Messages establish the link between two events, a send event and the corresponding receive event. This last machine conforms to the definitions of distributed execution and run. This corresponds to a distributed system with asynchronous point-to-point communication.

The page <http://hurault.perso.enseeiht.fr/MenagerieOfRefinements/> contains all the models discussed in this chapter². This page also gives indications to replay the proofs (mainly version of Rodin and SMT provers).

Events

The context **Event** introduces the set **EVENT**. The machine **Events** simply states that new events happen. The variable **past** holds past events, to ensure that no event occurs twice.

```

MACHINE A_Events
SEES A_Event
VARIABLES
  past // Set of events that have happened
INVARIANTS
  Tpast: past ∈ ℙ(EVENT)
EVENT INITIALISATION THEN past := ∅
EVENT happen
  ANY e // New event
  WHERE
    grd1: e ∈ EVENT \ past
  THEN
    act1: past := past ∪ {e}
END

```

```

CONTEXT A_Event
SETS
  EVENT
END

```

²The names of the models are of the form **L_Name**, where the prefix is used so that models are displayed in their refinement order in Rodin. The text of the chapter often omits this prefix but the extracts of the models retain it for reference.

Links between events

The machine **Communication** introduces the notion of link between an event and its cause. This link will be later realised with an explicit message where the send of a message will be a cause, and the reception will be the consequence. A new variable **links** is introduced. As this chapter deals with point-to-point communication, the following invariants are expected:

INVARIANTS

Tlinks: $\text{links} \in \text{past} \leftrightarrow \text{past}$
 inv0: $\text{dom}(\text{links}) \cap \text{ran}(\text{links}) = \emptyset$ // Causes (emissions) and consequences (receptions) are distinct.
 inv1: $\forall e_1, e_2, c \cdot e_1 \mapsto c \in \text{links} \wedge e_2 \mapsto c \in \text{links} \Rightarrow e_1 = e_2$ // A consequence has only one direct cause.
 inv2: $\forall e, c_1, c_2 \cdot e \mapsto c_1 \in \text{links} \wedge e \mapsto c_2 \in \text{links} \Rightarrow c_1 = c_2$ // A cause has at most one consequence.

The event **happen** is refined into two new events: an event **create** which is exactly **happen** and a new event **link** to establish a link.

```

EVENT create REFINES happen extended
... // unchanged

EVENT link REFINES happen // A new event occurs, with a link to its cause
ANY e // New event
  cause // Cause event
WHERE
  grd1:  $e \in \text{EVENT} \setminus \text{past}$ 
  grd2:  $\text{cause} \in \text{past}$  // The cause event has already happened.
  grd3:  $\text{cause} \notin \text{dom}(\text{links})$  // A cause serves once only.
  grd4:  $\text{cause} \notin \text{ran}(\text{links})$  // A cause is not a consequence.
THEN
  act1:  $\text{past} := \text{past} \cup \{e\}$ 
  act2:  $\text{links} := \text{links} \cup \{\text{cause} \mapsto e\}$ 
END

```

Distributed Executions

The context **Peer** extends **Event** with **PEER**, a set of peer identity.

The machine **DistributedExecution** adds two variables: **peerOf** that holds, for each event, the peer on which it has occurred, and **prec** which is the causality relation between events (also previously noted \prec_c in the text). The main invariants state that **prec** is a partial order, is totally ordered on peers, and contains **links**.

INVARIANTS

Tprec: $\text{prec} \in \text{past} \leftrightarrow \text{past}$
 TpeerOf: $\text{peerOf} \in \text{past} \rightarrow \text{PEER}$
 inv1: $(\text{past} \triangleleft \text{id}) \subseteq \text{prec}$ // prec is reflexive.
 inv2: $\text{prec} ; \text{prec} \subseteq \text{prec}$ // prec is transitive.
 inv3: $\text{prec} \cap \text{prec}^{-1} \subseteq \text{id}$ // prec is anti-symmetric.
 inv4: $\text{links} \subseteq \text{prec}$ // prec contains (the reflexive transitive closure of) links.
 inv5: $\forall e_1, e_2 \cdot e_1 \in \text{past} \wedge e_2 \in \text{past} \wedge \text{peerOf}(e_1) = \text{peerOf}(e_2) \Rightarrow e_1 \mapsto e_2 \in \text{prec} \vee e_2 \mapsto e_1 \in \text{prec}$
 // Events occurring on the same peer are totally ordered.

```

EVENT link REFINES link extended
ANY
  ...
  p
WHERE
  ...
  +grd5: p ∈ PEER
THEN
  ...
  +act3: peerOf := peerOf ∪ {e ↦ p}
           // The new event is causally after all events from the same peer (third line)
           // and after all events that causally precedes the cause (fourth line).
  +act4: prec :=
           prec
           ∪ {e ↦ e}
           ∪ {ep · ep ∈ past ∧ (∃ ep2 · ep2 ∈ past ∧ peerOf(ep2) = p ∧ ep ↦ ep2 ∈ prec) | ep ↦ e}
           ∪ {ep · ep ∈ past ∧ ep ↦ cause ∈ prec | ep ↦ e}
END

```

(The event **create** is similar, without the fourth line of the assignment **+act4** which is the transitive closure of **prec** with regard to **cause**)

Run

The next refinement is the machine **Run** which introduces a variable **run** as a total order of events and verifies that it is a linear extension of **prec**. The Event-B expression $e1 \mapsto e2 \in \text{run}$ is also noted $e1 <_{\sigma} e2$ in the text.

```

INVARIANTS
  Trun: run ∈ past  $\Leftrightarrow$  past
  inv0:  $\forall e1, e2 \cdot e1 \in \text{past} \wedge e2 \in \text{past} \Rightarrow e1 \mapsto e2 \in \text{run} \vee e2 \mapsto e1 \in \text{run}$  // run is total.
  inv1:  $(\text{past} \triangleleft \text{id}) \subseteq \text{run}$  // run is reflexive (consequence of inv0).
  inv2:  $\text{run} ; \text{run} \subseteq \text{run}$  // run is transitive.
  inv3:  $\text{run} \cap \text{run}^{-1} \subseteq \text{id}$  // run is anti-symmetric.
  inv4:  $\text{prec} \subseteq \text{run}$  // run extends prec.
  inv5:  $\forall e1, e2 \cdot e1 \in \text{past} \wedge e2 \in \text{past} \wedge \text{peerOf}(e1) = \text{peerOf}(e2)$ 
            $\Rightarrow ((e1 \mapsto e2 \in \text{prec}) \Leftrightarrow (e1 \mapsto e2 \in \text{run}))$ 

```

run is constructed by a superposition to both **create** and **link** actions with:

<pre> EVENT create REFINES create extended ... +act5: run := run ∪ {e ↦ e} ∪ {ep · ep ∈ past ep ↦ e} </pre>	<pre> EVENT link REFINES link extended ... +act5: run := run ∪ {e ↦ e} ∪ {ep · ep ∈ past ep ↦ e} </pre>
---	---

Messages

At last, messages are introduced. A message links its send event and its reception event. The machine **RunWithMessages** adds two variables **mesOf** and **comOf** which, for each event, specify the associated message and event type (send or receive). Conversely, the variable **links** and the parameter **cause** of event **link** (refined in **receive**) are removed. A gluing invariant **Glinks** and adequate witnesses in **receive** are provided.

This machine models a message-passing distributed execution with asynchronous point-to-point communication.

MACHINE E_RunWithMessages

REFINES D_Run

SEES E_Messages

VARIABLES

past
 peerOf
 prec
 run
 mesOf // Message mesOf(e) of a communication event e.
 comOf // label comOf(e) (Send or Receive)
 // of a communication event e.

CONTEXT E_Messages

EXTENDS C_Peers

SETS

MESSAGE
 COM

CONSTANTS

Send
 Receive

AXIOMS

COM: partition(COM, {Send}, {Receive})

INVARIANTS

TcomOf: comOf \in past \rightarrow COM

TmesOf: mesOf \in past \rightarrow MESSAGE

inv1: $\forall e1, e2 \cdot e1 \in \text{past} \wedge e2 \in \text{past} \wedge \text{comOf}(e1) = \text{comOf}(e2) \wedge \text{mesOf}(e1) = \text{mesOf}(e2) \Rightarrow e1 = e2$
 // no message is sent or received more than once

inv2: $\forall e \cdot e \in \text{past} \wedge \text{comOf}(e) = \text{Receive}$
 $\Rightarrow (\exists es \cdot es \in \text{past} \wedge \text{comOf}(es) = \text{Send} \wedge \text{mesOf}(e) = \text{mesOf}(es) \wedge es \mapsto e \in \text{prec})$
 // a receive event is preceded by a send event

Glinks: $\forall es, er \cdot es \mapsto er \in \text{links} \Rightarrow \text{comOf}(es) = \text{Send} \wedge \text{comOf}(er) = \text{Receive} \wedge \text{mesOf}(es) = \text{mesOf}(er)$

EVENT send **REFINES** create extended

ANY e // New event.

p // Peer where the event occurs.

m // Sent message.

WHERE

...
 grd3: $m \in \text{MESSAGE} \setminus \text{ran}(\text{mesOf})$

THEN

...
 +act5: $\text{mesOf} := \text{mesOf} \cup \{e \mapsto m\}$
 +act6: $\text{comOf} := \text{comOf} \cup \{e \mapsto \text{Send}\}$

EVENT receive **REFINES** link

ANY e // New event.

p // Receiver.

m // Received message.

WHERE

grd1: $e \in \text{EVENT} \setminus \text{past}$

grd4: $p \in \text{PEER}$

grd5: $m \in \text{MESSAGE}$

grd6: $\forall ep \cdot ep \in \text{past} \wedge \text{comOf}(ep) = \text{Receive} \Rightarrow \text{mesOf}(ep) \neq m$ // m has not been received yet.

grd7: $\exists es \cdot es \in \text{past} \wedge \text{comOf}(es) = \text{Send} \wedge \text{mesOf}(es) = m$ // m has been sent.

WITH

cause: $\text{cause} \in \text{past} \wedge \text{comOf}(\text{cause}) = \text{Send} \wedge \text{mesOf}(\text{cause}) = m$

links' : $\text{links}' = \text{links} \cup \{es \cdot es \in \text{past} \wedge \text{comOf}(es) = \text{Send} \wedge \text{mesOf}(es) = m \mid es \mapsto e\}$

THEN

act1: $\text{past} := \text{past} \cup \{e\}$

act3: $\text{peerOf} := \text{peerOf} \cup \{e \mapsto p\}$

act4: $\text{prec} := \text{prec} \cup \{e \mapsto e\}$

$\cup \{ep \cdot ep \in \text{past} \wedge (\exists ep2 \cdot ep2 \in \text{past} \wedge \text{peerOf}(ep2) = p \wedge ep \mapsto ep2 \in \text{prec}) \mid ep \mapsto e\}$
 $\cup \{ep \cdot ep \in \text{past} \wedge (\exists es \cdot es \in \text{past} \wedge \text{comOf}(es) = \text{Send} \wedge \text{mesOf}(es) = m \wedge ep \mapsto es \in \text{prec}) \mid ep \mapsto e\}$

act5: $\text{run} := \text{run} \cup \{e \mapsto e\} \cup \{ep \cdot ep \in \text{past} \mid ep \mapsto e\}$

+act6: $\text{mesOf} := \text{mesOf} \cup \{e \mapsto m\}$

+act7: $\text{comOf} := \text{comOf} \cup \{e \mapsto \text{Receive}\}$

6.2.4 Summary

The initial model may seem contrived with no invariant and a trivial event. It forms the foundation for the refinements that, at each step, introduce an essential element up to a model of message-passing distributed execution with point-to-point asynchronous communication. The machine **Communication** introduces point-to-point communication with a relation between two events. By playing with the guards, other models could be specified: imposing that consequences are also causes (synchronous communication); allowing several consequences for a cause (multicast); allowing several causes for a consequence (join). The next machine, **DistributedExecution**, introduces peers and the causality relation (a partial order, total on events on the same peers). Then, the machine **Run** introduces an observation of a distributed execution as a total order on events. Lastly **RunWithMessages** identifies the relation between two events as a message and expresses the two essential invariants on messages (uniqueness of reception and reception after send). This last machine models a fully asynchronous point-to-point communication model.

6.3 Abstract Communication Models

The interaction model (or communication model) plays a major role in distributed systems. It specifies when a communication action (send or receive) is possible in order to ensure specific properties on the communication. For instance, a globally ordered communication model imposes that all messages are received in the same order as they were sent: no message overtakes another on a run.

The seven communication models from Chapter 2 and their relations are studied below. Their specifications are expressed as properties on distributed executions or runs. They serve as invariants on Event-B machines that specify the distributed executions permitted by each model. An Event-B machine is built for each communication model. Refinement is used to define a hierarchy based on simulation. The strengthening of the delivery order reduces the non-determinism of the receptions.

6.3.1 Specifications of the Communication Models

We consider the specifications of the communication models with events. Each communication model is characterised by an invariant that describes the ordering properties it ensures on the communication. The model-specific invariants are presented in Table 6.1. For *RSC*, the invariant states that no event (e) can occur between the send event of a message (es) and the receive event of this message (er). The common part for the four FIFO models and the *Causal* model introduces es_1 and es_2 as send events of two distinct messages, er_1 and er_2 as the corresponding receive events. The specific part imposes an order on the receive events based on the causal or run order of es_1 and es_2 , and on the equality of the sending and/or receiving peers (same sending peer and same receiving peer for *FIFO 1-1*, same sending peer for *FIFO 1-n*, same receiving peer for *FIFO n-1* and *Causal*). For instance, the ordering invariant in the machine **Fifo11Event** is:

$$\begin{aligned}
& \forall es1, er1, es2, er2 \cdot es1 \in \text{past} \wedge er1 \in \text{past} \wedge es2 \in \text{past} \wedge er2 \in \text{past} \\
& \wedge \text{comOf}(es1) = \text{Send} \wedge \text{comOf}(es2) = \text{Send} \\
& \wedge \text{comOf}(er1) = \text{Receive} \wedge \text{comOf}(er2) = \text{Receive} \\
& \wedge \text{mesOf}(es1) = \text{mesOf}(er1) \wedge \text{mesOf}(es2) = \text{mesOf}(er2) \\
& \wedge \text{peerOf}(es1) = \text{peerOf}(es2) \wedge \text{peerOf}(er1) = \text{peerOf}(er2) \\
& \wedge es1 \mapsto es2 \in \text{prec} \\
& \Rightarrow er1 \mapsto er2 \in \text{run}
\end{aligned}$$

Model	Invariant of the Communication Model
async	\top
RSC	$\forall es, er.$ $es \in past \wedge er \in past$ $\wedge comOf(es) = Send \Rightarrow \neg(\exists e. e \in past$ $\wedge comOf(er) = Receive \wedge es <_{\sigma} e$ $\wedge mesOf(es) = mesOf(er) \wedge e <_{\sigma} er)$
common to fifo* and causal	$Common \triangleq \left(\begin{array}{l} es_1 \in past \wedge es_2 \in past \\ \wedge er_1 \in past \wedge er_2 \in past \\ \wedge comOf(es_1) = Send \\ \wedge comOf(es_2) = Send \\ \wedge comOf(er_1) = Receive \\ \wedge comOf(er_2) = Receive \\ \wedge mesOf(es_1) = mesOf(er_1) \\ \wedge mesOf(es_2) = mesOf(er_2) \end{array} \right)$
fifo11	$\forall es_1, es_2, er_1, er_2.$ $\left(\begin{array}{l} Common \\ \wedge es_1 \prec_c es_2 \\ \wedge peerOf(es_1) = peerOf(es_2) \\ \wedge peerOf(er_1) = peerOf(er_2) \end{array} \right) \Rightarrow er_1 <_{\sigma} er_2$
causal	$\forall es_1, es_2, er_1, er_2.$ $\left(\begin{array}{l} Common \\ \wedge es_1 \prec_c es_2 \\ \wedge peerOf(er_1) = peerOf(er_2) \end{array} \right) \Rightarrow er_1 <_{\sigma} er_2$
fifo1n	$\forall es_1, es_2, er_1, er_2.$ $\left(\begin{array}{l} Common \\ \wedge es_1 \prec_c es_2 \\ \wedge peerOf(es_1) = peerOf(es_2) \end{array} \right) \Rightarrow er_1 <_{\sigma} er_2$
fifon1	$\forall es_1, es_2, er_1, er_2.$ $\left(\begin{array}{l} Common \\ \wedge es_1 <_{\sigma} es_2 \\ \wedge peerOf(er_1) = peerOf(er_2) \end{array} \right) \Rightarrow er_1 <_{\sigma} er_2$
fifonn	$\forall es_1, es_2, er_1, er_2.$ $\left(\begin{array}{l} Common \\ \wedge es_1 <_{\sigma} es_2 \end{array} \right) \Rightarrow er_1 <_{\sigma} er_2$

Table 6.1: Model-specific invariants in the Event-B machines with distributed executions. The common part for the fifo* and causal models introduces es_1 and es_2 as send events, er_1 and er_2 as the corresponding receive events. The specific part imposes an order on the receive events based on the causal or run order of es_1 and es_2 , and on the equality of the sending and/or receiving peers. The Weakest Preconditions of these invariants are used as guards on the *send* and *receive* events.

It states that if two messages are sent from the same peer (events $es1$ and $es2$) and are received on the same peer (events $er1$ and $er2$), and that $es1$ occurs before $es2$ (on their common peer), then the events $er1$ and $er2$ must occur in this order in the run. One noteworthy point of these specifications is that they implicitly include message loss as a never-delivered message: the corresponding receive event of a send event does not exist in the generated run.

Our next goal is to compare the communication models (Section 6.3.2) by proving that some have less transitions than others (i.e. are more deterministic). Later in Sections 6.4 and 6.5, we derive more concrete models. However, at the current point, it is important to have machines that are as liberal as the ordering allows. Thus, the weakest preconditions of the ordering invariants are stipulated for the send and receive events and are used as guards on both communication events. As the actions are assignments of the form $var := var \cup \{\dots\}$, the computation of the weakest preconditions is trivial [DS90]. As an example, Figure 6.2 shows the machine describing the *FIFO 1-1* communication model.

6.3.2 Reduction of Non-Determinism

The communication models presented in the previous chapters constitute gradual steps between fully asynchronous distributed communication (*Fully Asynchronous*) where sending and receiving a message is always possible, partially ordered communication (*FIFO 1-1*, *Causal*, *FIFO 1-n*, *FIFO n-1*), totally ordered communication (*FIFO n-n*), and almost synchronous communication (*RSC*) where a message must be received immediately after it has been sent.

The machine `RunWithMessages` models asynchronous communication and corresponds to the *Fully Asynchronous* model. The other models impose more and more determinism on reception (and, for *RSC*, on send). The hierarchy is thus proved by refinement. Note however that these are *not* concretisation refinements: no model can be called more (or less) concrete or realisable. Concretisation of the communication models will follow a specific path for each model and is described in Section 6.4 and Section 6.5.

The results of the formal proofs of refinement between the Event-B machines are summed up in Figure 6.3. The two machines for *FIFO n-n* are actually the same (strictly same context, same variables and same events): one refines *FIFO 1-n*, the other one refines *FIFO n-1*. This propagates to the two machines for *RSC*.

6.3.3 Proofs and Invariants

The difference between the communication models is an invariant directly related to the order of delivery, and the associated weakest precondition used as a guard on the communication events. A proof of refinement consists in proving the logical implications between these invariants. Most of the time these proofs require little manual intervention thanks to auto-provers, post-tactics, and SMT solvers.

The refinements of *Causal* in *FIFO n-1* and *FIFO 1-n* need manual intervention with the help of a specific invariant:

$$\begin{aligned}
& \forall e1, e2 \cdot e1 \mapsto e2 \in \text{prec} \wedge \text{peerOf}(e1) \neq \text{peerOf}(e2) \\
& \Rightarrow (\exists es, er \cdot e1 \mapsto es \in \text{prec} \\
& \quad \wedge es \mapsto er \in \text{prec} \wedge er \mapsto e2 \in \text{prec} \\
& \quad \wedge \text{peerOf}(e1) = \text{peerOf}(es) \\
& \quad \wedge \text{comOf}(es) = \text{Send} \\
& \quad \wedge \text{comOf}(er) = \text{Receive} \\
& \quad \wedge \text{mesOf}(es) = \text{mesOf}(er))
\end{aligned}$$

```

MACHINE G2_Fifo11Event
REFINES E_RunWithMessages
SEES E_MESSAGES
VARIABLES // unchanged
    past
    peerOf
    prec
    run
    mesOf
    comOf

EVENT send REFINES send extended
    ANY e
        p
        m
    WHERE
        grd1: e ∈ EVENT \ past
        grd2: p ∈ PEER
        grd3: m ∈ MESSAGE \ ran(mesOf)
        // weakest precondition of the fifo11 ordering invariant
        ordering: ∀ es1, er1, es2, er2 · es1 ∈ past ∪ {e} ∧ er1 ∈ past ∪ {e}
            ∧ es2 ∈ past ∪ {e} ∧ er2 ∈ past ∪ {e}
            ∧ (comOf ∪ {e ↦ Send})(es1) = Send ∧ (comOf ∪ {e ↦ Send})(es2) = Send
            ∧ (comOf ∪ {e ↦ Send})(er1) = Receive ∧ (comOf ∪ {e ↦ Send})(er2) = Receive
            ∧ (mesOf ∪ {e ↦ m})(es1) = (mesOf ∪ {e ↦ m})(er1)
            ∧ (mesOf ∪ {e ↦ m})(es2) = (mesOf ∪ {e ↦ m})(er2)
            ∧ (peerOf ∪ {e ↦ p})(es1) = (peerOf ∪ {e ↦ p})(es2)
            ∧ (peerOf ∪ {e ↦ p})(er1) = (peerOf ∪ {e ↦ p})(er2)
            ∧ es1 ↦ es2 ∈ run ∪ {e ↦ e} ∪ {ep · ep ∈ past | ep ↦ e}
            ⇒ er1 ↦ er2 ∈ run ∪ {e ↦ e} ∪ {ep · ep ∈ past | ep ↦ e}
    THEN // unchanged
        act1: past := past ∪ {e}
        act2: peerOf := peerOf ∪ {e ↦ p}
        act3: prec := prec ∪ ...
        act4: run := run ∪ ...
        act5: mesOf := mesOf ∪ {e ↦ m}
        act6: comOf := comOf ∪ {e ↦ Send}

EVENT receive REFINES receive extended
    ...
    WHERE
        ...
        // weakest precondition of the fifo11 ordering invariant
        ordering: ∀ es1, er1, es2, er2 · es1 ∈ past ∪ {e} ∧ er1 ∈ past ∪ {e}
            ∧ es2 ∈ past ∪ {e} ∧ er2 ∈ past ∪ {e}
            ∧ (comOf ∪ {e ↦ Receive})(es1) = Send ∧ (comOf ∪ {e ↦ Receive})(es2) = Send
            ∧ (comOf ∪ {e ↦ Receive})(er1) = Receive ∧ (comOf ∪ {e ↦ Receive})(er2) = Receive
            ∧ (mesOf ∪ {e ↦ m})(es1) = (mesOf ∪ {e ↦ m})(er1)
            ∧ (mesOf ∪ {e ↦ m})(es2) = (mesOf ∪ {e ↦ m})(er2)
            ∧ (peerOf ∪ {e ↦ p})(es1) = (peerOf ∪ {e ↦ p})(es2)
            ∧ (peerOf ∪ {e ↦ p})(er1) = (peerOf ∪ {e ↦ p})(er2)
            ∧ es1 ↦ es2 ∈ run ∪ {e ↦ e} ∪ {ep · ep ∈ past | ep ↦ e}
            ⇒ er1 ↦ er2 ∈ run ∪ {e ↦ e} ∪ {ep · ep ∈ past | ep ↦ e}
    THEN
        ... // unchanged

```

Figure 6.2: *FIFO 1-1* Communication Model, described with events. Weakest preconditions are used to ensure that the events are no more constrained than the ordering invariant that defines the communication model.

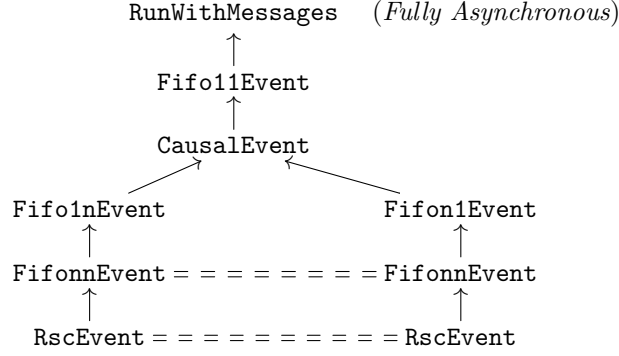


Figure 6.3: Refinements between the Event-B machines of the communication models. An arrow means “refines”. The lines of “=” express that both **FifonnEvent** models are strictly the same, except for different parent, and the same for **RscEvent**.

This invariant states that two causally related events on different peers are necessarily linked by (at least) one message. Informally, it means that causality between events on distinct peers only exists only due to message exchanges. This invariant is verified in the machine **RunWithMessages** but is only needed when refining the machine **CausalEvent**. During the proof, it had to be manually instantiated.

6.4 History-based Communication Models

In this section, we present a first set of refinements of the communication models. This step shares a common framework based on the concept of message histories which allows to compare them. Yet, the models are realistic enough, in the sense that they could be implemented and used as such. The ordering properties are realised by keeping track of dependant messages in histories. Refinement is used in two ways: to verify that a machine that models a communication model with histories refines the corresponding machine where the communication model is described by a relation between events (Section 6.4.2); and to prove that the hierarchy of Figure 6.3 is preserved (Section 6.4.3).

6.4.1 Specifications with Histories

We consider specifications of the asynchronous point-to-point interaction models where communication occurs according to two parameterised events:

Action	Parameters	Description
send	$p \in \text{PEER}, d \in \text{PEER}, m \in \text{MESSAGE}$	peer p sends message m to peer d
receive	$p \in \text{PEER}, m \in \text{MESSAGE}$	peer p receives message m

In order to simplify the specifications and refinements, unlike the communication actions in Chapter 4, the destination peers are here explicit parameters. We therefore do not use channels and this also means a given message will always reach a deterministic destination. Since the completeness of some models in the framework in Chapter 4 require mono-receptor systems, this choice in this chapter for the menagerie of refinement is not a significant setback.

The models rely on a state variable **net** that contains messages in transit. Messages are labelled to carry information about the communication: the origin peer, the destination peer, and

the history of the message. The history of a message is the set of messages on which it depends, i.e. the set of messages which precede it. As two notions of precedence exist (causal/execution), two kinds of message histories are defined: namely causal and global.

Definition 39 (Message Histories). *For a run $\sigma = (E, \prec_c, <_\sigma, com, mes, peer)$,*

$$\forall m \in MESSAGE :$$

$$hgOf(m) \triangleq \left\{ m' \in MESSAGE : \exists e, e' \in E : \begin{array}{l} com(e) = Send \wedge com(e') = Send \\ \wedge mes(e) = m \wedge mes(e') = m' \\ \wedge e' <_\sigma e \end{array} \right\}$$

$$hcOf(m) \triangleq \left\{ m' \in MESSAGE : \exists e, e' \in E : \begin{array}{l} com(e) = Send \wedge com(e') = Send \\ \wedge mes(e) = m \wedge mes(e') = m' \\ \wedge e' \prec_c e \end{array} \right\}$$

In the Event-B models, the message histories are built upon state variables $hg \subseteq MESSAGE$, the global history and $hc \in PEER \rightarrow \mathbb{P}(MESSAGE)$, the causal histories of each peer. When peer p sends a message m , the global history ($hgOf$) and the causal history ($hcOf$) of m are the current values of hg and of $hc(p)$. The new message is also added to the history state variables (hg and $hc(p)$). The causal history $hc(p)$ of a destination peer p is updated when a message m is received to account for the causal relation induced by the transmission of the message from one peer to another: m and its causal history $hcOf(m)$ are added to $hc(p)$. The ordering properties of a model are determined by specific guards on the **send** and **receive** events that depend on the message histories, origin, and destination of a message.

6.4.2 Refinement of Events by Histories

For each communication model, the refinement of the event-based model in the history-based model is done in two steps. The first step adds the new variables to hold histories and message destination (**net**, **hg**, **hc**, **hgOf**, **hcOf**, **destOf**), and replaces the guards on events by guards on histories. Then, the second step removes the now useless variables related to events (**past**, **prec**, **run**...). Thus, for *FIFO 11*, we have **MACHINE** F2_Fifo11History **REFINES** E_Fifo11Event and **MACHINE** G2_Fifo11History **REFINES** F2_Fifo11History. The machine F2_Fifo11History combines events and histories, and the machine G2_Fifo11History is the cleaned up machine. Doing this in two steps significantly facilitates the proofs.

The resulting model for *FIFO 1-1* is shown in Figure 6.4. The ordering invariants for each model are presented in Table 6.2 and the ordering guards are in Table 6.3. For instance, the *FIFO 1-1* ordering invariant states that if m_1 and m_2 have the same origin and destination, and that m_1 was sent before m_2 (thus m_1 is in the causal history of m_2), then m_1 cannot be still in transit while m_2 has already been received. This means that m_1 must be received before m_2 . The ordering guard for receive allows to deliver a message m if there does not exist another message m_2 in transit, with same origin and destination, and which is in the history of m .

Gluing and Additional Invariants

The machine that specifies a communication model based on events is refined into a machine where the state variables, type invariants, event parameters, guards, and actions dealing with message histories are added. The main proof obligations consist in proving that the model-specific guards on each communication event (see Table 6.3) suffice to guarantee the ordering properties on the distributed executions (see Table 6.1). This is achieved with an ordering invariant stated

MACHINE G2_Fifo11History
REFINES F2_Fifo11History
SEES E_Messages

VARIABLES

net // Network
hg // Global history
hc // Causal history per peer
origOf // sender of message
destOf // destination of message
hgOf // global history of message
hcOf // causal history of message

INVARIANTS

Tnet: net $\in \mathbb{P}(hg)$
Thg: hg $\in \mathbb{P}(\text{MESSAGE})$
Thc: hc $\in \text{PEER} \rightarrow \mathbb{P}(hg)$
TorigOf: origOf $\in hg \rightarrow \text{PEER}$
TdestOf: destOf $\in hg \rightarrow \text{PEER}$
ThgOf: hgOf $\in hg \rightarrow \mathbb{P}(hg)$
ThcOf: hcOf $\in hg \rightarrow \mathbb{P}(hg)$
ordering: $\forall m1, m2 \cdot m1 \in hg \wedge m2 \in hg \wedge m1 \neq m2$
 $\wedge \text{origOf}(m1) = \text{origOf}(m2)$
 $\wedge \text{destOf}(m1) = \text{destOf}(m2)$
 $\wedge m1 \in \text{hcOf}(m2)$
 $\Rightarrow \neg (m1 \in \text{net} \wedge m2 \notin \text{net})$

EVENT INITIALISATION THEN

a1: net := \emptyset
...
a9: hcOf := \emptyset

EVENT send **REFINES** send

ANY p m d

WHERE

Tp: p $\in \text{PEER}$
Tm: m $\in \text{MESSAGE} \setminus hg$
 // new message id
Td: d $\in \text{PEER}$

THEN

a1: net := net $\cup \{m\}$
a2: hg := hg $\cup \{m\}$
a4: hc(p) := hc(p) $\cup \{m\}$
a5: origOf := origOf $\cup \{m \mapsto p\}$
a6: destOf := destOf $\cup \{m \mapsto d\}$
a7: hgOf := hgOf $\cup \{m \mapsto hg\}$
a8: hcOf := hcOf $\cup \{m \mapsto \text{hc}(p)\}$

EVENT receive **REFINES** receive

ANY p m

WHERE

Tp: p $\in \text{PEER}$
intransit : m $\in \text{net}$
destination : destOf(m) = p

THEN

a1: net := net $\setminus \{m\}$
a2: hc(p) := hc(p) $\cup \text{hcOf}(m) \cup \{m\}$

Figure 6.4: History-Based Event-B Model for Fifo 11 Communication

<i>Fully Asynchronous</i>	\top
<i>FIFO 1-1</i>	$\begin{aligned} &\forall m_1, m_2 \cdot m_1 \in \text{hg} \wedge m_2 \in \text{hg} \wedge m_1 \neq m_2 \\ &\quad \wedge \text{origOf}(m_1) = \text{origOf}(m_2) \wedge \text{destOf}(m_1) = \text{destOf}(m_2) \wedge m_1 \in \text{hcOf}(m_2) \\ &\Rightarrow \neg (m_1 \in \text{net} \wedge m_2 \notin \text{net}) \end{aligned}$
<i>Causal</i>	$\begin{aligned} &\forall m_1, m_2 \cdot m_1 \in \text{hg} \wedge m_2 \in \text{hg} \wedge m_1 \neq m_2 \\ &\quad \wedge \text{destOf}(m_1) = \text{destOf}(m_2) \wedge m_1 \in \text{hcOf}(m_2) \\ &\Rightarrow \neg (m_1 \in \text{net} \wedge m_2 \notin \text{net}) \end{aligned}$
<i>FIFO 1-n</i>	$\begin{aligned} &\forall m_1, m_2 \cdot m_1 \in \text{hg} \wedge m_2 \in \text{hg} \wedge m_1 \neq m_2 \\ &\quad \wedge \text{origOf}(m_1) = \text{origOf}(m_2) \wedge m_1 \in \text{hcOf}(m_2) \\ &\Rightarrow \neg (m_1 \in \text{net} \wedge m_2 \notin \text{net}) \end{aligned}$
<i>FIFO n-1</i>	$\begin{aligned} &\forall m_1, m_2 \cdot m_1 \in \text{hg} \wedge m_2 \in \text{hg} \wedge m_1 \neq m_2 \\ &\quad \wedge \text{destOf}(m_1) = \text{destOf}(m_2) \wedge m_1 \in \text{hgOf}(m_2) \\ &\Rightarrow \neg (m_1 \in \text{net} \wedge m_2 \notin \text{net}) \end{aligned}$
<i>FIFO n-n</i>	$\forall m_1, m_2 \cdot m_1 \in \text{hg} \wedge m_2 \in \text{hg} \wedge m_1 \neq m_2 \wedge m_1 \in \text{hgOf}(m_2) \Rightarrow \neg (m_1 \in \text{net} \wedge m_2 \notin \text{net})$
<i>RSC</i>	$\text{net} = \emptyset \vee (\exists m \cdot \text{net} = \{m\})$

Table 6.2: Ordering Invariants for the History-based Communication Models. For instance, Fifo11 invariant states that if m_1 and m_2 have the same origin and destination, and that m_1 was sent before m_2 (thus m_1 is in the causal history of m_2), then m_1 cannot be still in transit while m_2 has already been received. This means that m_1 must be received before m_2 . Likewise, the other invariants state the ordering of the models. RSC simply states that there is at most one message in transit.

Communication Model	Guard for $\mathbf{send}(p,d,m)$	Guard for $\mathbf{receive}(p,m)$
<i>Fully Async.</i>	\top	$m \in net \wedge destOf(m) = p$
<i>FIFO 1-1</i>	\top	$m \in net$ $\wedge destOf(m) = p$ $\wedge \neg(\exists m_2 \cdot m_2 \in net \wedge origOf(m) = origOf(m_2)$ $\wedge destOf(m_2) = p$ $\wedge m_2 \in hcOf(m))$
<i>Causal</i>	\top	$m \in net$ $\wedge destOf(m) = p$ $\wedge \neg(\exists m_2 \cdot m_2 \in net \wedge destOf(m_2) = p$ $\wedge m_2 \in hcOf(m))$
<i>FIFO 1-n</i>	\top	$m \in net$ $\wedge destOf(m) = p$ $\wedge \neg(\exists m_2 \cdot m_2 \in net \wedge origOf(m) = origOf(m_2)$ $\wedge m_2 \in hcOf(m))$
<i>FIFO n-1</i>	\top	$m \in net$ $\wedge destOf(m) = p$ $\wedge \neg(\exists m_2 \cdot m_2 \in net \wedge destOf(m_2) = p$ $\wedge m_2 \in hgOf(m))$
<i>FIFO n-n</i>	\top	$m \in net$ $\wedge destOf(m) = p$ $\wedge \neg(\exists m_2 \cdot m_2 \in net \wedge m_2 \in hgOf(m))$
<i>RSC</i>	$net = \emptyset$	$m \in net \wedge destOf(m) = p$

Table 6.3: Model-specific guards in the Event-B machines with message histories. The common guards deal with types.

INVARIANTS

Ghg: $hg = \text{ran}(\text{mesOf})$
 Gnet: $\forall m \cdot m \in hg \wedge \neg (\exists e2 \cdot e2 \in \text{past} \wedge \text{comOf}(e2) = \text{Receive} \wedge \text{mesOf}(e2) = m) \Rightarrow m \in \text{net}$
 Gnet2: $\forall e \cdot e \in \text{past}$
 $\wedge \text{comOf}(e) = \text{Send}$
 $\wedge \neg (\exists e2 \cdot e2 \in \text{past} \wedge \text{comOf}(e2) = \text{Receive} \wedge \text{mesOf}(e2) = \text{mesOf}(e))$
 $\Rightarrow \text{mesOf}(e) \in \text{net}$
 Gnet3: $\forall e \cdot e \in \text{past} \wedge \text{comOf}(e) = \text{Receive} \Rightarrow \text{mesOf}(e) \notin \text{net}$
 GorigOf: $\forall e \cdot e \in \text{past} \wedge \text{comOf}(e) = \text{Send} \Rightarrow \text{origOf}(\text{mesOf}(e)) = \text{peerOf}(e)$
 GdestOf: $\forall e \cdot e \in \text{past} \wedge \text{comOf}(e) = \text{Receive} \Rightarrow \text{destOf}(\text{mesOf}(e)) = \text{peerOf}(e)$
 GhgOf: $\forall es1, es2 \cdot es1 \mapsto es2 \in \text{run}$
 $\wedge es1 \neq es2$
 $\wedge \text{comOf}(es2) = \text{Send}$
 $\Rightarrow \text{mesOf}(es1) \in hgOf(\text{mesOf}(es2))$
 GhgOf2: $\forall e1, e2, m1, m2 \cdot m1 \in hg \wedge m2 \in hg \wedge m1 \neq m2 \wedge e1 \mapsto e2 \in \text{run}$
 $\wedge \text{comOf}(e1) = \text{Send} \wedge \text{comOf}(e2) = \text{Send}$
 $\wedge \text{mesOf}(e1) = m1 \wedge \text{mesOf}(e2) = m2$
 $\Rightarrow m1 \in hgOf(m2)$
 GorigOf2: $\forall m \cdot m \in hg \Rightarrow \text{origOf}(m) \in \{e \cdot e \in \text{past} \wedge \text{comOf}(e) = \text{Send} \wedge \text{mesOf}(e) = m \mid \text{peerOf}(e)\}$
 GhcOf: $\forall es1, es2 \cdot es1 \mapsto es2 \in \text{prec}$
 $\wedge es1 \neq es2$
 $\wedge \text{comOf}(es2) = \text{Send}$
 $\Rightarrow \text{mesOf}(es1) \in hcOf(\text{mesOf}(es2))$
 GhcOf2: $\forall e1, e2, m1, m2 \cdot m1 \in hg \wedge m2 \in hg \wedge m1 \neq m2 \wedge e1 \mapsto e2 \in \text{prec}$
 $\wedge \text{comOf}(e2) = \text{Send} \wedge \text{mesOf}(e1) = m1 \wedge \text{mesOf}(e2) = m2$
 $\Rightarrow m1 \in hcOf(m2)$
 Ghc: $\forall e1, e2 \cdot e1 \mapsto e2 \in \text{prec} \Rightarrow \text{mesOf}(e1) \in hc(\text{peerOf}(e2))$
 Hinv1: $\text{net} \subseteq hg$
 Hinv2: $\forall m \cdot m \in hg \Rightarrow m \notin hgOf(m)$
 Hinv3: $\forall m \cdot m \in hg \Rightarrow m \notin hcOf(m)$
 Hinv4: $\forall m, m2 \cdot m \in hg \wedge m2 \notin hgOf(m) \Rightarrow m2 \notin hcOf(m)$
 fifo11_ordering: $\forall m1, m2 \cdot m1 \in hg \wedge m2 \in hg$
 $\wedge \text{origOf}(m1) = \text{origOf}(m2)$
 $\wedge \text{destOf}(m1) = \text{destOf}(m2)$
 $\wedge m1 \in hcOf(m2)$
 $\Rightarrow \neg (m1 \in \text{net} \wedge m2 \notin \text{net})$

Figure 6.5: Common invariants involved in the proofs of refinement between the models using events and the models using histories. They include invariants on the communication models using histories (prefix: H) along with gluing invariants that link the state variables from both models (prefix: G). The invariant `fifo11_ordering` is specific to the machine `Fifo11History`.

on histories (see Table 6.2) and additional invariants to make the glue between the event-based model and the history-based model.

Some invariants that are common to all the communication models have been highlighted to prove the refinements. They are presented in Figure 6.5. The first group of invariants are gluing invariants between the state variables of the concrete machine (network, histories, and the associated accessors) and the state variables of the abstract machine (distributed executions). To make the proofs automatic, it was sometimes necessary to state the same (or similar) invariant in different ways. The three invariants **Gnet**, **Gnet2**, **Gnet3** express that a message m is in transit ($m \in \mathbf{net}$) iff the message has been sent ($m \in \mathbf{hg}$, or there exists a send event for it) and there is no receive event for it. **GhgOf** and **GhgOf2** are close invariants which state that a message m_1 is in the global history of m_2 if m_1 has been sent before (in the sense of total order $<_\sigma$, coded in **run**) message m_2 . Likewise, invariants **GhcOf** and **GhcOf2** relate causal history and causal order $<_c$ (coded in **prec**).

Removal of Useless Variables

Once the refinement of an event-based model by a history-based model has been done, the variables related to events (**past**, **prec**, **run**, and the accessors **peerOf**, **mesOf**, **comOf**) are no longer necessary. A second refinement is used to hide them. This refinement depends on a bijection between **EVENT** and **COM** \times **MESSAGE**. This bijection means that events (in the sense of distributed execution, Definition 1) are uniquely defined by an action (send or receive) on a given message. This bijection is introduced in the context **MESSAGES**:

CONTEXT E_MESSAGES CONSTANTS ... eventmapper AXIOMS T1: eventmapper $\in (\mathbf{COM} \times \mathbf{MESSAGE}) \mapsto \mathbf{EVENT}$
--

The final **G2_Fifo11History** is thus **F2_Fifo11History**, stripped of the variables related to events, and with a witness to replace the removed parameter:

MACHINE G2_Fifo11History REFINES F2_Fifo11History ... INVARIANTS eventmapper: eventmapper[$\{\text{Send}\} \times \mathbf{hg}$] \cup eventmapper[$\{\text{Receive}\} \times (\mathbf{hg} \setminus \mathbf{net})$] = past EVENT send REFINES send ANY p m d WITH e: e = eventmapper(Send \mapsto m) ... EVENT receive REFINES receive ANY p m WITH e: e = eventmapper(Receive \mapsto m) ...
--

6.4.3 Preservation of the Hierarchy

Each history-based model is a refinement of its corresponding event-based model (for instance **Fifo11History** refines **Fifo11Event**), which proves that it obeys the ordering rule of the ab-

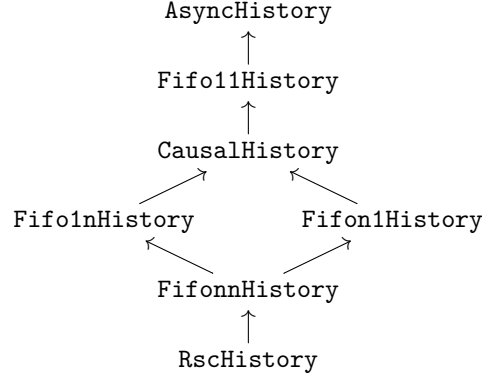


Figure 6.6: Refinements between the Event-B machines of the history-based communication models. An arrow means “refines”. As in Figure 6.3, there are two versions of **FifonnHistory**, strictly identical except for their parent, and they are merged in the figure.

stract communication model based on events. As explained in Section 6.3.2, the communication models also form a hierarchy based on their non-deterministic nature (e.g. **CausalEvent** refines **Fifo11Event**, see Figure 6.3). This same hierarchy is preserved with history-based models (Figure 6.6).

The history-based approach provides descriptions of the communication models with a similar template. For a given communication model, only one (or none) kind of history is actually useful. *FIFO 1-1*, *Causal* and *FIFO 1-n* rely only on the causal message histories, *FIFO n-n* and *FIFO n-1* rely on the global message histories, and *Fully Asynchronous* and *RSC* on none at all. Nevertheless, experience revealed that when it comes to proving the refinements between the models, keeping trace of all the histories in all the models was more convenient and sometimes necessary in practice. This way, only guards on the **send** (*RSC*) and **receive** (other models) events differ between the models, while the actions themselves are the same.

Since only a guard on the **receive** event (**send** in the case of *RSC*) differs between the communication models, proving the refinement between two models consists in proving that one guard implies the other. The guards have quite similar structures: they specify that no message in transit is in the (global or causal) history of the one that is to be retrieved, and additional conditions about senders or destinations can be involved (Table 6.3). Therefore, the proofs do not require complex manual interventions. They mostly rely on the invariants of Figure 6.5, specifically the **Hinv*** ones, and are mainly first-order implications with quantifiers derived from the ordering invariants (i.e. *causal_ordering* \Rightarrow *fifo11_ordering*).

The refinement from **CausalHistory** to **Fifo1nHistory** is the trickiest. It requires to highlight a specific property of *fifo1n*: messages in transit that are causally related have necessarily the same origin peer (this property is false for causal/fifo11/asynchronous communication). Invariant **i2** below formalises this property and **i1** is a prerequisite to its proof. This property is crucial to the proof of refinement.

MACHINE Fifo1nHistory INVARIANTS

// Required to prove that *Fifo1nHistory* refines *CausalHistory*.

i1: $\forall m, p \cdot m \in \text{net} \wedge p \in \text{PEER} \wedge m \in \text{hc}(p) \Rightarrow \text{origOf}(m) = p$

i2: $\forall m1, m2 \cdot m1 \in \text{net} \wedge m2 \in \text{net} \wedge m2 \in \text{hcOf}(m1) \Rightarrow \text{origOf}(m1) = \text{origOf}(m2)$

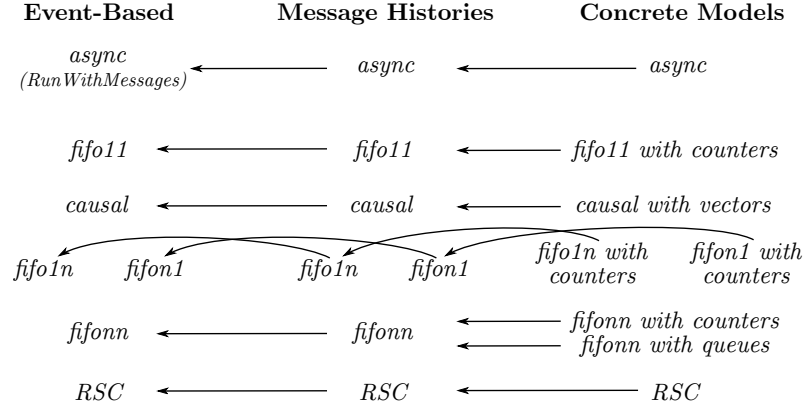


Figure 6.7: Refinements between the Event-B machines of the different approaches for each communication model. An arrow means “refines”. The *async* and *RSC* concrete implementations are no different from their message histories counterparts.

6.5 Concrete Communication Models

Two descriptions of the communication models introduced in Chapter 2 have been provided in 6.3.1 and 6.4.1. The abstract models based on events directly translate the ordering policies of the communication models. They are high level models where the distributed aspect of the interaction between peers is abstracted by a run that contains communication events. The second approach of modelling using message histories is more concrete: the locality and transmission of data is taken into account with messages that carry their history. However, keeping trace of all the previously sent messages is still unrealistic in practice. Therefore we refine the models that use histories with concrete models using counters of messages or queues of messages. The proposed concrete models are shown in Figure 6.7. They serve as a proof of concept of how one can refine a high level specification of a communication model into concrete models.

6.5.1 Refinement with Counters of Messages

Depending on the degree of distribution of the communication model, if n denotes the number of peers, 2 counters (*FIFO n-n*), $2 \times n$ counters (*FIFO n-1* and *FIFO 1-n*), or $2 \times n^2$ counters (*FIFO 1-1*) are used to account for the ordinal rank of the last sent and last received messages in the system (*FIFO n-n*), a peer (*FIFO n-1* and *FIFO 1-n*), or couple of peers (*FIFO 1-1*). The rank of the last sent message serves to associate the rank to a new message at send. The ranks of the last received messages determine the rank of the messages that can be received (possibly depending on origin and destination). The Event-B machine of *FIFO 1-1* with counters and the invariants that have been highlighted to achieve the proof of refinement are shown below. The gluing invariant **GrankOf2** shows that counters are consistent with causality (for messages that have same origin and destination). The three invariants **inv*** show the relations between the counters and the ranks of the messages.

MACHINE H2_Fifo11Counter
REFINES G2_Fifo11History
SEES E__Messages

VARIABLES

...
rankOf // Accessor: ordinal rank of a message
lastReceived // Rank of the last received message
lastSent // Rank of the last sent message

INVARIANTS

...
TrankOf: rankOf \in net $\rightarrow \mathbb{N}$
TlastSent: lastSent \in (PEER \times PEER) $\rightarrow \mathbb{N}$
TlastReceived: lastReceived \in (PEER \times PEER) $\rightarrow \mathbb{N}$
GrankOf: $\forall m_1, m_2 \cdot m_1 \in \text{net} \wedge m_2 \in \text{net}$
 $\wedge \text{origOf}(m_1) = \text{origOf}(m_2)$
 $\wedge \text{destOf}(m_1) = \text{destOf}(m_2)$
 $\wedge m_1 \neq m_2$
 $\Rightarrow \text{rankOf}(m_1) \neq \text{rankOf}(m_2)$
GrankOf2: $\forall m_1, m_2 \cdot m_1 \in \text{net} \wedge m_2 \in \text{net}$
 $\wedge m_1 \in \text{hcOf}(m_2)$
 $\wedge \text{origOf}(m_1) = \text{origOf}(m_2)$
 $\wedge \text{destOf}(m_1) = \text{destOf}(m_2)$
 $\Rightarrow \text{rankOf}(m_1) < \text{rankOf}(m_2)$
inv1: $\forall m \cdot m \in \text{net} \Rightarrow \text{lastReceived}(\text{origOf}(m) \mapsto \text{destOf}(m)) < \text{rankOf}(m)$
inv2: $\forall m \cdot m \in \text{net} \Rightarrow \text{rankOf}(m) \leq \text{lastSent}(\text{origOf}(m) \mapsto \text{destOf}(m))$
inv3: $\forall p_1, p_2 \cdot p_1 \in \text{PEER} \wedge p_2 \in \text{PEER} \Rightarrow \text{lastReceived}(p_1 \mapsto p_2) \leq \text{lastSent}(p_1 \mapsto p_2)$

EVENT INITIALISATION THEN

...
+act5: rankOf := \emptyset
+act6: lastReceived := {pp·pp \in PEER \times PEER | pp \mapsto 0}
+act7: lastSent := {pp·pp \in PEER \times PEER | pp \mapsto 0}

EVENT send **REFINES** send

ANY p m d **WHERE**
grd1: p \in PEER
grd2: m \in MESSAGE \hg // new message id
grd3: d \in PEER
THEN
act1: net := net \cup {m}
act2: hg := hg \cup {m}
act3: destOf := destOf \cup {m \mapsto d}
act4: origOf := origOf \cup {m \mapsto p}
+act5: rankOf := rankOf \cup {m \mapsto 1 + lastSent(p \mapsto d)}
+act6: lastSent (p \mapsto d) := 1 + lastSent(p \mapsto d)

EVENT receive **REFINES** receive

ANY p m **WHERE**
grd1: p \in PEER
grd2: m \in net
grd3: destOf(m) = p
ordering: rankOf(m) = lastReceived(origOf(m) \mapsto p) + 1
THEN
act1: net := net \ {m}
+act2: rankOf := {m} \ll rankOf
+act3: lastReceived (origOf(m) \mapsto p) := rankOf(m)

6.5.2 Refinement with Queues of Messages

Alternatively, message queues (FIFO) can be used instead, hence the names of the communication models if n denotes the number of peers: a global queue (*FIFO* n - n), n inbox queues (*FIFO* n -1) or outbox queues (*FIFO* 1- n), or n^2 queues (*FIFO* 1-1). Queues³ are functions from $0..n$ to MESSAGE. The context **Queues** provides axioms and theorems used in the proof of refinement between *FIFO* n - n with message histories and *FIFO* n - n with queues. The gluing invariant **Gluing2** states that if m_1 is in the global history of m_2 , then m_1 occurs before m_2 in the queue.

```

CONTEXT I__Queues EXTENDS E__Messages
CONSTANTS
  Queue size append head tail
AXIOMS
  axm1: Queue  $\in \mathbb{P}(\mathbb{N} \rightarrow \text{MESSAGE})$ 
  axm2: Queue =  $\{\emptyset\} \cup \text{Union}(\{n \cdot n \in \mathbb{N} \mid 0 \dots n \rightarrow \text{MESSAGE}\})$ 
  thm1:  $\forall q \cdot q \in \text{Queue} \Rightarrow \text{finite}(q)$ 
  axm3: size  $\in \text{Queue} \rightarrow \mathbb{N}$ 
  axm4:  $\forall q \cdot q \in \text{Queue} \Rightarrow (\exists n \cdot n \in \mathbb{N} \wedge \text{dom}(q) = 0 \dots n \wedge \text{size}(q) = n)$ 
  axm5: append  $\in \text{MESSAGE} \times \text{Queue} \rightarrow \text{Queue}$ 
  axm6:  $\forall q, m \cdot q \in \text{Queue} \wedge m \in \text{MESSAGE} \Rightarrow \text{append}(m \mapsto q) = q \cup \{\text{size}(q) + 1 \mapsto m\}$ 
  axm7: head  $\in \text{Queue} \setminus \{\emptyset\} \rightarrow \text{MESSAGE}$ 
  axm8:  $\forall q \cdot q \in \text{Queue} \setminus \{\emptyset\} \Rightarrow \text{head}(q) = q(0)$ 
  axm9: tail  $\in \text{Queue} \setminus \{\emptyset\} \rightarrow \text{Queue}$ 
  axm10:  $\forall q \cdot q \in \text{Queue} \setminus \{\emptyset\} \Rightarrow \text{tail}(q) = \{i \cdot i \in 0 \dots \text{size}(q) - 1 \mid i \mapsto q(i + 1)\}$ 

```

```

MACHINE I6a_FifonnQueue
REFINES G6a_FifonnHistory
SEES I__Queues

VARIABLES
  hg
  destOf
  queue // shared FIFO

INVARIANTS
  Tqueue: queue  $\in \text{Queue}$ 
  Gluing1: ran(queue) = net
  Gluing2:  $\forall m1, m2 \cdot m1 \in \text{net} \wedge m2 \in \text{net} \wedge m1 \in \text{hgOf}(m2) \Rightarrow (\exists i, j \cdot i \in \mathbb{N} \wedge j \in \mathbb{N} \wedge i < j \wedge \text{queue}(i) = m1 \wedge \text{queue}(j) = m2)$ 

```

```

EVENT send REFINES send
ANY p m d WHERE
  ...
THEN
  act1: hg := hg  $\cup \{m\}$ 
  act2: destOf := destOf  $\cup \{m \mapsto d\}$ 
  +act3: queue := append(m  $\mapsto$  queue)

EVENT receive REFINES receive
ANY p m WHERE
  grd1: p  $\in \text{PEER}$ 
  grd2: m  $\in \text{ran}(\text{queue}) \wedge \text{destOf}(m) = p$ 
  order: queue  $\neq \emptyset \wedge m = \text{head}(\text{queue})$ 
THEN
  +act1: queue := tail(queue)

```

6.5.3 Logical Clocks

Regarding the causal model, the causality relation can be explicit, using pruned causal histories [KS98] (in the worst case, this is as costly as our version with histories), or derived from logical vector clocks of size n [Fid88, Mat89] or matrix clocks of size $n \times n$ [RST91, Ray13].

Every peer p has a vector clock $vcOf(p)$. For peers p and pp , $vcOf(p)(pp)$ holds the number of events on pp that are in the current past of peer p . When a peer sends a message, it increments

³We have also modelled queues with sequences (Seq) from the *BasicTheory* of the Theory Plugin [BM13] but we have chosen to dispense with this plugin as our usage was simple.

its own count ($vcOf(p)(p)$) and piggybacks its current vector with the message. When a peer receives a message, it updates every component of its vector with the max of its current value and of the component of the received vector. Thus, $vcOf(p)(pp)$ holds the number of messages sent by pp and known by p . A message m is in the causal history of m' if and only if every vector component of m is lower or equal than the one of m' (and at least one is strictly lower: no two different messages can have the same vector). To ensure causal reception, a message can be delivered to a peer if and only if no other message exists for this peer with a lower timestamp.

The refinement of **CausalHistory** replaces the history variables (**hc**, **hcOf**) with vector clocks (**vcOf**, **rankOf**). The events are refined to update these variables and, in the **receive** event, the guard built on histories is replaced with a property on the vectors. A similar derivation is presented in [YB05].

```
MACHINE H3_CausalVector
REFINES G3_CausalHistory
SEES E_Messages
VARIABLES
  net
  hg
  origOf
  destOf
  rankOf // message →
vector clock
  vcOf // peer → vector clock
INVARIANTS
  TrankOf: rankOf ∈ net →
  (PEER → ℕ)
  TvcOf: vcOf ∈ PEER →
  (PEER → ℕ)
```

```
EVENT send REFINES send
ANY p m d WHERE
  Tm: m ∈ MESSAGE \ hg
  Tpd: p ∈ PEER ∧ d ∈ PEER
THEN
  act1: net := net ∪ {m}
  act2: hg := hg ∪ {m}
  act3: origOf := origOf ∪ {m ↦ p}
  act4: destOf := destOf ∪ {m ↦ d}
  act5: vcOf(p) := vcOf(p) ⧹ {p ↦ vcOf(p)(p)+1}
  act6: rankOf(m) := vcOf(p) ⧹ {p ↦ vcOf(p)(p)+1}
```

```
EVENT receive REFINES receive
ANY p m WHERE
  Tm: m ∈ net
  Tp: p ∈ PEER
  dest: destOf(m) = p
  order: ¬(∃ m2 · m2 ∈ net \ {m}
    ∧ destOf(m2) = p
    ∧ (∀ pp · pp ∈ PEER ⇒ rankOf(m2)(pp) ≤ rankOf(m)(pp)))
THEN
  act1: net := net \ {m}
  act2: vcOf(p) := { pp · pp ∈ PEER | pp ↦ max({ vcOf(p)(pp), rankOf(m)(pp) }) }
  act3: rankOf := {m} ⧹ rankOf
END
```

The refinement proof requires gluing invariants on causal histories and vector clocks. Invariant **inv1** relates the timestamps of two messages and states that if message $m1$ is in the causal history of $m2$, then every vector component of $m1$ is lower or equal than the one of $m2$. Invariant **inv2** exhibits the vector component that is certainly strictly lower. Invariant **inv3** relates the timestamps of messages and peers.

```
INVARIANTS
  inv1: ∀ m1, m2 · m1 ∈ net ∧ m2 ∈ net ∧ m1 ≠ m2
    ⇒ (m1 ∈ hcOf(m2) ⇒ (∀ p · p ∈ PEER ⇒ rankOf(m1)(p) ≤ rankOf(m2)(p)))
  inv2: ∀ m1, m2 · m1 ∈ net ∧ m2 ∈ net ∧ m1 ∈ hcOf(m2)
    ⇒ rankOf(m1)(origOf(m2)) < rankOf(m2)(origOf(m2))
  inv3: ∀ m, p · m ∈ net ∧ p ∈ PEER ∧ m ∈ hc(p)
    ⇒ (∀ pp · pp ∈ PEER ⇒ rankOf(m)(pp) ≤ vcOf(p)(pp))
```

6.6 Additional Remarks

6.6.1 Proof Effort

The menagerie holds 42 machines, 41 refinements, 329 invariants, and more than 1400 proof obligations. Once the necessary invariants are stated, the large majority of these proof obligations are automatically proved by Rodin with SMT solvers (49 manual proofs, 3.5% of the proof obligations). The main difficulties are described below.

- To make the proofs automatic, the trick is to find additional invariants. For instance, to prove that **RscHistory** refines **RscEvent**, the following invariant has to be made explicit (it says that if there exists at least one event after a send event e_1 , then the message sent at e_1 is no longer in transit).

$$\forall e_1, e_2 \cdot e_1 \mapsto e_2 \in \text{run} \wedge \text{comOf}(e_1) = \text{Send} \wedge e_1 \neq e_2 \Rightarrow \text{mesOf}(e_1) \notin \text{net}$$

As expected, the discovery of the necessary invariants was the hardest part in the proofs, and the majority of the proof efforts was devoted to this point. Our methodology was to run the automatic provers, and to analyse the failure (if any). After some case analysis of the disjunctions, a contradiction often appeared in the hypotheses. This contradiction leads us to a relevant new invariant. Once stated and proved, this new invariant may, with good luck, suppress the unsuccessful branch and advance towards the fully automatic proof.

- The refinements of *Causal* in *FIFO n-1* and *FIFO 1-n* are never easy. One essential invariant states that:

$$\begin{aligned} & \forall e_1, e_2 \cdot e_1 \mapsto e_2 \in \text{prec} \wedge \text{peerOf}(e_1) \neq \text{peerOf}(e_2) \\ & \Rightarrow (\exists es, er \cdot e_1 \mapsto es \in \text{prec} \wedge es \mapsto er \in \text{prec} \wedge er \mapsto e_2 \in \text{prec} \\ & \quad \wedge \text{comOf}(es) = \text{Send} \wedge \text{comOf}(er) = \text{Receive} \\ & \quad \wedge \text{peerOf}(e_1) = \text{peerOf}(es) \\ & \quad \wedge \text{mesOf}(es) = \text{mesOf}(er)) \end{aligned}$$

It states that two causally related events on different peers are necessary linked by (at least) one message, or conversely, that causality between different peers only arises from message exchanges. Nevertheless, this invariant has to be manually instantiated.

- For each communication model, the Receive/GRD obligation of the refinement of the event-based model by the history-based model requires light manual intervention. The proof mainly consists in case disjunctions of the four events that appear in the ordering invariant (Are they equal? Is one of them the new receive event?) and instantiating the relevant guard.
- The refinements which eliminate events, once unused, need witnesses that rely on a bijection between **EVENT** and **COM** \times **MESSAGE** (**eventmapper**). Proofs often require to abstract away the expressions where it occurs.
- Concrete models need ad-hoc reasoning. For instance, Section 6.5.3 presents the specific invariants that are required to prove that **CausalVector** refines **CausalHistory**. These invariants are expected as they state that vector clocks encode causality. Nevertheless, the refinement proofs require to manually recall and instantiate these invariants.

6.6.2 Deadlock Freedom

In all the communication models (except *RSC*), the send event is always enabled. Thus all these models are trivially deadlock free. Actually, a stronger property is expected: if there are messages in transit, then at least one of them can be received. This means that (at least) a receive event (with adequate parameters) must be enabled if *net* is not empty. This is still a weak liveness property, as it does not state that all sent messages are eventually received: if there are finitely many send events, all messages must eventually be received, but if there are infinitely many send events, some messages may never be received without invalidating this property.

This property is stated as an invariant which has the form $net \neq \emptyset \Rightarrow \text{guard of receive}$ (see Table 6.3). For instance, in the model *G2_Fifo11History* (Figure 6.4), the invariant is:

$$net \neq \emptyset \Rightarrow (\exists m, p \cdot p \in \text{PEER} \wedge m \in net \wedge \text{destOf}(m) = p \\ \wedge \neg(\exists m2 \cdot m2 \in net \\ \wedge \text{origOf}(m) = \text{origOf}(m2) \\ \wedge \text{destOf}(m2) = p \\ \wedge m2 \in \text{hcOf}(m)))$$

and in the model *H2_Fifo11Counter* (Section 6.5.1), the invariant is:

$$net \neq \emptyset \Rightarrow (\exists p, m \cdot p \in \text{PEER} \\ \wedge m \in net \\ \wedge \text{destOf}(m) = p \\ \wedge \text{rankOf}(m) = \text{lastReceived}(\text{origOf}(m) \mapsto p) + 1)$$

The model checker ProB has been used to check this invariant with a small number of peers and messages (3 or 4 depending on the model).

6.6.3 Previous Work in TLA^+

In [CHQ16], we have presented TLA^+ [Lam02] specifications for the seven communication models. The goals of this other work were different. A first objective was to develop a framework to verify systems [CHQ15]. In this framework, systems are described with TLA^+ actions, and the communication model and the properties of correctness can easily be changed thanks to TLA^+ modules. A second objective was to compare the communication models. As in this chapter, the communication models are described with events and with message histories. However only the communication model with message histories are specified in TLA^+ . The same hierarchy of the communication models is proved in [CHQ16]. For the event-based models, the proofs were paper-proofs and used inclusion properties on orders, whereas refinement is used in this chapter. The hierarchy of the history-based models in TLA^+ was proved by refinement using TLAPS [CHMQ16]. Event-B refinements offer a similar approach in this chapter. [CHQ16] was paying attention to both the correctness and the completeness of the models: do the history-based models conform to the event-based models? Are the history-based models as liberal as possible, allowing the same set of executions as the event-based models? In the Event-B models, refinement ensures correctness, and the use of weakest preconditions provide completeness.

6.6.4 Utility of the Hierarchies

These hierarchies of refinement are important for substitutability, the ability to replace one model with another. If a communication model M_1 refines a communication model M_2 , it means that

M_1 cannot deliver more messages than M_2 , or conversely, that any message that M_1 delivers is also deliverable in M_2 . Thus, for any system using asynchronous point-to-point communication, a safety property that is proved with M_2 is necessarily true when substituting M_2 with M_1 . Of course, liveness properties are not guaranteed to be preserved, as the refined model allows fewer behaviors. For instance, RSC does not allow two consecutive send events without a receive event between them, and thus a system may deadlock with RSC while progressing with a more liberal model.

This substitutability is linked to model decomposition. A distributed application is refined up to the point where communication is introduced. Once components and communication are apparent, a model decomposition can be used to isolate the communication part. The hierarchy of communication models is then used to choose an adequate ordered communication model, and the concretisation of this model is a step toward implementation.

There exist three existing approaches to decomposition in Event-B [HISW11]: shared-variable [AH07], shared-event [But09, Sil11] and modularisation [ITL⁺10]. All three focus on allowing independent refinements of sub-models while ensuring that the recomposition of the refined sub-models is a refinement of the original model. In shared-variable decomposition, events of a model are partitioned in several sub-models and a part of the state (the shared variables) is replicated in several sub-models. In shared-event decomposition, variables are partitioned and a set of events are synchronised and shared by sub-models. In modularisation, interfaces composed of variables and operations are specified for sub-components. Operations are specified by pre/post-conditions, hiding their implementation.

Our menagerie of refinements seems well suited for shared-event decomposition. During the refinement of a system, asynchronous communication appears via two events send and receive. These events are isolated in a sub-model to be further refined using the results of this chapter. We have already followed a similar approach in TLA⁺ to develop the framework for the automated verification of asynchronous communicating systems [CHQ15] in Chapter 4 and Chapter 5: the communication model and the peers (which contain the application part) are connected via a synchronous product on communication events. Note that, as observed in [HISW11, p. 10], while shared-event decomposition naturally leads to CSP/CCS-like synchronous communicating systems, introducing a “buffer” (in our case the variable `net`) allows to model asynchronous communication.

6.6.5 Localisation

The last point concerns the distributed nature of the communication models. The first abstract models, based on properties of the executions, are purely logical and offer a global point of view of the communication models. The second models, based on histories, are actually directly implementable even if costly. By looking at Table 6.3, one can distinguish distributed communication models and centralised communication models. The distributed communication models (*Fully Asynchronous*, *FIFO 1-1*, *Causal*, *FIFO 1-n*) only need meta information piggybacked with the message and local knowledge available on the peer. The centralised communication models (*FIFO n-1*, *FIFO n-n*, *RSC*) require global shared knowledge. The lower part of the hierarchy contains the centralised communication models, and the upper part are the distributed ones.

This consideration applies to localisation. In the four communication models *Fully Asynchronous*, *FIFO 1-1*, *Causal*, *FIFO 1-n*, be it history-based or concrete, the variables⁴ which occur

⁴The remaining variables in these models are `hg` and `net`. `hg` is solely used to uniquely identify messages and could trivially be refined in peer-local counters `PEER×N` as long as peers are uniquely identified. The variable `net` models the transport layer which could be further refined for instance with communication channels as in

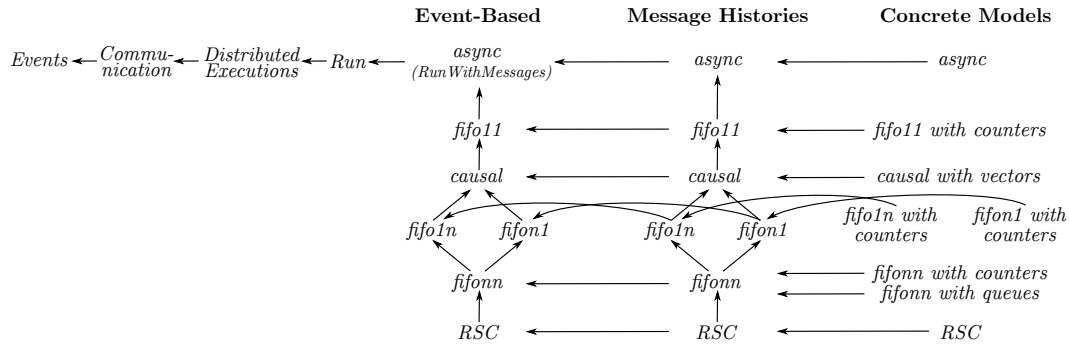


Figure 6.8: All the Proved Refinements between the Event-B machines. An arrow means “refines”.

in the ordering predicate are associated to a message (`origOf`, `destOf`, `hcOf` in the history-based models or `rankOf` in the concrete `H2_Fifo11counter` and `H3_CausalVector`), or are indexed by peer (`hc` in the history-based models, `vcOf` in `CausalVector`, `lastSent` and `lastReceived` in `Fifo11Counter`...). In contrast, the models *FIFO n-1*, *FIFO n-n*, *RSC* are based on globally shared variables (for instance `hg` in `Fifo11History` or `queue` in `FifonnQueue`). This comes from their definition which depends on the run order ($<_{\sigma}$) of independent events occurring on distinct peers. The implementation in a distributed system requires a central coordinator or a totally ordered multicast.

6.7 Conclusion

This chapter considers the diversity of point-to-point asynchronous communication models with different approaches for their modelling in Event-B. It starts with abstract models based on distributed executions such as in Chapter 2, it then considers a uniform modelling using message histories that is similar to the specifications of the communication models in Chapter 4, and then presents concrete models. The relations between the models are refinements. The overall picture of the proved refinements is shown in Figure 6.8.

A methodology to model the communication part of a distributed system uses both directions of refinement:

1. Use the hierarchy to find the most adequate / required communication model;
2. Use the concretisation refinements to find the most concrete / less expensive model.

In the first step, a communication model is chosen, either beforehand, or when verifying the under-development system. The hierarchy helps in choosing the degree of determinism of the communication model. The abstract specifications (Table 6.1) give intelligible ordering properties, and the specifications based on histories (Table 6.3) provide a model that can be used to study and prove the considered system. Then, different versions of the retained communication model can be used. The concretisation refinements allow to substitute a model for another.

Ongoing work aims at extending the menagerie with new models, introducing broadcast or multicast (analogous to a message consumed by more than one peer) and join (a synchronised reception of a set of messages). Moreover, broadcast possesses new orderings that are independent of the ordering of the send events, namely totally ordered broadcast, where different messages are

[ACM03].

received in the same order on different peers. A rich enough modelling needs to take into account the notion of group. A broadcast targets a group of recipients, and ordering is defined w.r.t to the groups: local total order (messages to the same group); pair-wise total order (at the intersection of the recipient groups); global total order (independently of the groups). Another question regards round-based algorithms. In round-based algorithms, a computation is divided in rounds. At round n , messages from round $n - 1$ are received, a local action on each peer is done, and messages are sent for round $n + 1$. This computation model, called synchronous in the distributed systems community, allows to solve consensus in presence of node failure. However, it is less strict than synchronous communication as in CCS/CSP (or than *Realisable with Synchronous Communication*), allowing several in transit messages and asynchronous communication inside a round.

Part III

Group Communication

Chapter 7

On the Particularities of Multicast Communication

The formal base for the study of multicast communication (or group communication) is laid out in Chapter 2. Although the seven communication models studied throughout this work remain the same in both point-to-point and multicast communication, the latter reveals to lead to more complex modelling, analysis, and results. For example, the hierarchy between the different communication models happens to be simpler in point-to-point communication. While the three previous chapters provide an in-depth analysis of point-to-point communication, the particularities of the more generic multicast communication are at the heart of this chapter. The chapter aims at extending the work carried with point-to-point communication so it encompasses multicast communication. It is divided in two sections that respectively focus on structural and operational aspects of multicast communication.

The first section considers a message ordering property called total ordering that only makes sense in multicast communication along with a specific case of communication here called one-to-all communication. It compares both of them to the known communication models in order to enrich the existing hierarchy. The second section presents ongoing work on a TLA⁺ framework for the compatibility checking of compositions of peers that features and seamlessly integrates point-to-point and multicast communication. The conception choices are motivated and the limitations of these choices discussed along with potential solutions. Eventually, a recap of the current progress in the study of multicast communication concludes the chapter.

7.1 Extension of the Hierarchy of Communication Models

7.1.1 Totally Ordered Multicast Distributed Executions

Some distributed systems feature duplicated peers that are supposed to serve the same purpose and make the overall system more robust. A message that would be sent to a single peer in point-to-point communication is sent, in multicast communication, to all the duplicates. In such cases, it is interesting to guarantee that the same messages are received in the same order by all the duplicates. This way, the receptions of a message by all the duplicates may be viewed as atomic, as if the duplicates were abstracted by a single peer that receives the message in question. This property is called total ordering (not to be mistaken for the mathematical concept of total order) and is independent of other ordering policies (e.g. *FIFO 1-1*). It may simply be considered as

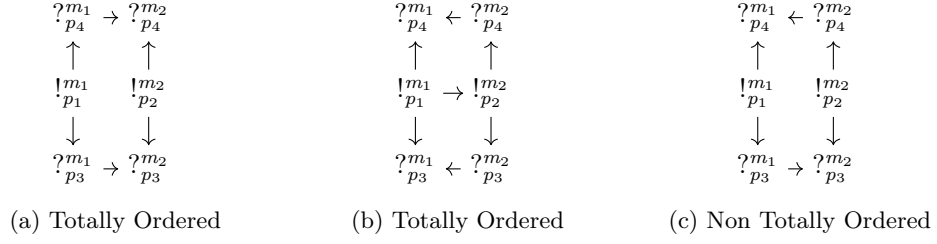


Figure 7.1: Examples of distributed executions $(E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \text{com}, \text{peer}, \text{mes})$ that illustrate total ordering. A send event of a message m by a peer p is denoted $!_p^m$ (an event $e \in E$ such that $\text{com}(e) = \text{Send} \wedge \text{mes}(e) = m \wedge \text{peer}(e) = p$). A receive event of a message m on a peer p is denoted $?_p^m$ (an event $e \in E$ such that $\text{com}(e) = \text{Receive} \wedge \text{mes}(e) = m \wedge \text{peer}(e) = p$). A path from e_1 to e_2 means $e_1 \prec_c e_2$.

another communication model and is usually combined with either the *Causal* communication model (total-causal) or *FIFO* communication (the combination with total ordering makes the *FIFO* models indistinguishable because total ordering propagates the *FIFO* ordering to all the peers). The following formalises this concept and updates the hierarchy established in Chapter 2 accordingly.

Definition 40 (Total Ordering). Exec^T is the set of distributed executions such that messages are received (when they are received) in the same order on all the peers. Such a distributed execution is said to be totally ordered.

$$\text{Exec}^T \triangleq \left\{ \left(\begin{array}{c} E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \\ \text{com}, \text{peer}, \text{mes} \end{array} \right) \in \text{Exec} \mid \begin{array}{l} \forall es_1, es_2, er_1, er_2, er'_1, er'_2 \in E : \\ \left(\begin{array}{l} \text{com}(es_1) = \text{Send} \\ \wedge \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{Receive} \\ \wedge \text{com}(er_2) = \text{Receive} \\ \wedge \text{com}(er'_1) = \text{com}(er'_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) = \text{mes}(er'_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) = \text{mes}(er'_2) \\ \wedge \text{peer}(er_1) = \text{peer}(er_2) \\ \wedge \text{peer}(er'_1) = \text{peer}(er'_2) \\ \wedge er_1 \prec_c er_2 \\ \Rightarrow er'_1 \prec_c er'_2 \end{array} \right) \end{array} \right\}$$

Run^T denotes the set of totally ordered runs that extend totally ordered distributed executions.

The distributed execution in Figure 7.1a is totally ordered because on peers p_3 and p_4 , m_1 and m_2 are received in the same order, a property that does not hold in the distributed execution in Figure 7.1c. Regarding the totally ordered distributed execution in Figure 7.1b, it is important to notice that, although there is a causal dependence between the two send events, the receptions can happen in the reverse order. Total ordering is another independent ordering policy that happens not to depend on the causality between the send events.

Theorem 41. *The FIFO n-1 runs are totally ordered.*

$$\text{Run}_{n-1} \subseteq \text{Run}^T$$

Proof. Let $\sigma \triangleq (E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \prec_\sigma, \text{com}, \text{peer}, \text{mes}) \in \text{Run}_{n1}$.

$$\text{Let } es_1, es_2, er_1, er_2, er'_1, er'_2 \in E \text{ such that: } \left(\begin{array}{l} \text{com}(es_1) = \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{com}(er_2) = \text{Receive} \\ \wedge \text{com}(er'_1) = \text{com}(er'_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) = \text{mes}(er'_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) = \text{mes}(er'_2) \\ \wedge \text{peer}(er_1) = \text{peer}(er_2) \\ \wedge \text{peer}(er'_1) = \text{peer}(er'_2) \\ \wedge er_1 \prec_c er_2 \end{array} \right)$$

Assume $\neg es_1 \prec_\sigma es_2$.

$es_1 \neq es_2$ and $es_2 \prec_\sigma es_1$ because \prec_σ is a linear order on E .

$er_2 \prec_c er_1$ because $\sigma \in \text{Run}_{n1}$. **Contradiction.**

Therefore $es_1 \prec_\sigma es_2$.

1. Case $es_1 = es_2$:

$er'_1 = er'_2$ because $\text{mes}(es_1) = \text{mes}(er'_1) = \text{mes}(es_2) = \text{mes}(er'_2)$, $\text{peer}(er_1) = \text{peer}(er_2)$ and $\sigma \in \text{Run}$ and a message is received at most once on a given peer by Definition 8 (page 26) of Run.

QED $er'_1 \prec_c er'_2$ by reflexivity.

2. Case $es_1 \neq es_2$:

QED $er'_1 \prec_c er'_2$ because $\sigma \in \text{Run}_{n1}$.

□

Corollary 42. *All the runs that are FIFO n-1, FIFO n-n, and RSC are totally ordered.*

$$\text{Run}_{RSC} \subseteq \text{Run}_{nn} \subseteq \text{Run}_{n1} \subseteq \text{Run}^T$$

Proof. From Theorem 17 (page 39) and the previous theorem. □

Figure 7.2 depicts the upgraded knowledge about the inclusion of the sets of runs. It shows how total ordering relates to the other ordering policies. Each area in the diagram has a label that corresponds to a witness run in the first column of Table 7.1 and Table 7.2.

7.1.2 One-to-All Communication

A specific case of multicast communication is one-to-all communication. Messages are received by *all* the peers in the distributed system, including the sender. This is not an ordering policy. In practice, reliable broadcast implies one-to-all communication.

Definition 43 (One-to-All Distributed Execution and Run). *A one-to-all distributed execution $(E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \text{com}, \text{peer}, \text{mes})$ is a multicast distributed execution where every sent message is received exactly once on each peer. The set of one-to-all distributed executions is denoted Exec^A .*

$$\text{Exec}^A \triangleq \left\{ (E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \text{com}, \text{peer}, \text{mes}) \in \text{Exec} \mid \begin{array}{l} \forall e \in E : \\ \text{com}(e) = \text{Send} \\ \Rightarrow \\ \left(\begin{array}{l} \forall p \in \mathcal{P} : \\ \exists e' \in E : \\ \left(\begin{array}{l} \text{com}(e') = \text{Receive} \\ \wedge \text{peer}(e') = p \\ \wedge \text{mes}(e') = \text{mes}(e) \end{array} \right) \end{array} \right) \end{array} \right\}$$

	Set	Example in Run	Example in Run^A
1	Run_{RSC}		
2	$\text{Run}_{nn} \setminus \text{Run}_{RSC}$		
3	$\left(\begin{matrix} \text{Run}_{1n} \\ \cap \\ \text{Run}_{n1} \end{matrix} \right) \setminus \text{Run}_{nn}$		
4	$\left(\begin{matrix} \text{Run}_{1n} \\ \cap \\ \text{Run}_c \\ \cap \\ \text{Run}^T \end{matrix} \right) \setminus \text{Run}_{n1}$		
5	$(\text{Run}_{1n} \cap \text{Run}^T) \setminus \text{Run}_c$		Impossible (Theorem 44)
6	$(\text{Run}_{1n} \cap \text{Run}_c) \setminus \text{Run}^T$		
7	$\text{Run}_{1n} \setminus (\text{Run}^T \cup \text{Run}_c)$		
8	$\text{Run}_{n1} \setminus \text{Run}_{1n}$		

Table 7.1: Example Runs Associated to the Numbered Labels in Figure 7.2

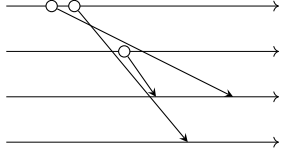
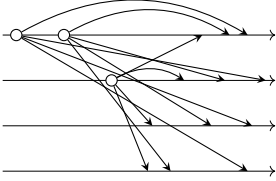
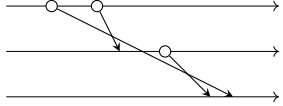
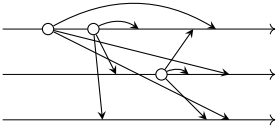
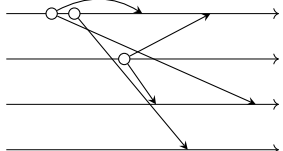
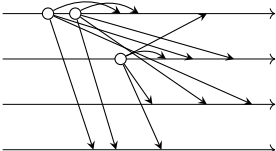
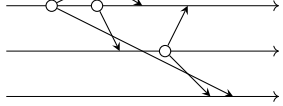
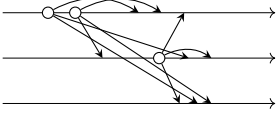

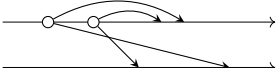
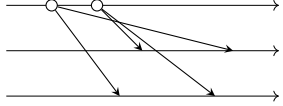
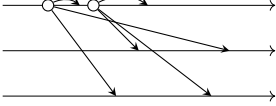
	Set	Example in Run	Example in Run ^A
9	$(\text{Run}_c \cap \text{Run}^T) \setminus (\text{Run}_{n1} \cup \text{Run}_{1n})$		
10	$(\text{Run}_{11} \cap \text{Run}^T) \setminus (\text{Run}_c \cup \text{Run}_{1n})$		
11	$\text{Run}_c \setminus (\text{Run}_{1n} \cup \text{Run}^T)$		
12	$\text{Run}_{11} \setminus \left(\begin{array}{c} \text{Run}_c \\ \cup \\ \text{Run}_{1n} \\ \cup \\ \text{Run}^T \end{array} \right)$		
13	$\text{Run}^T \setminus \text{Run}_{11}$		
14	$\text{Run} \setminus (\text{Run}_{11} \cup \text{Run}^T)$		

Table 7.2: Example Runs Associated to the Numbered Labels in Figure 7.2

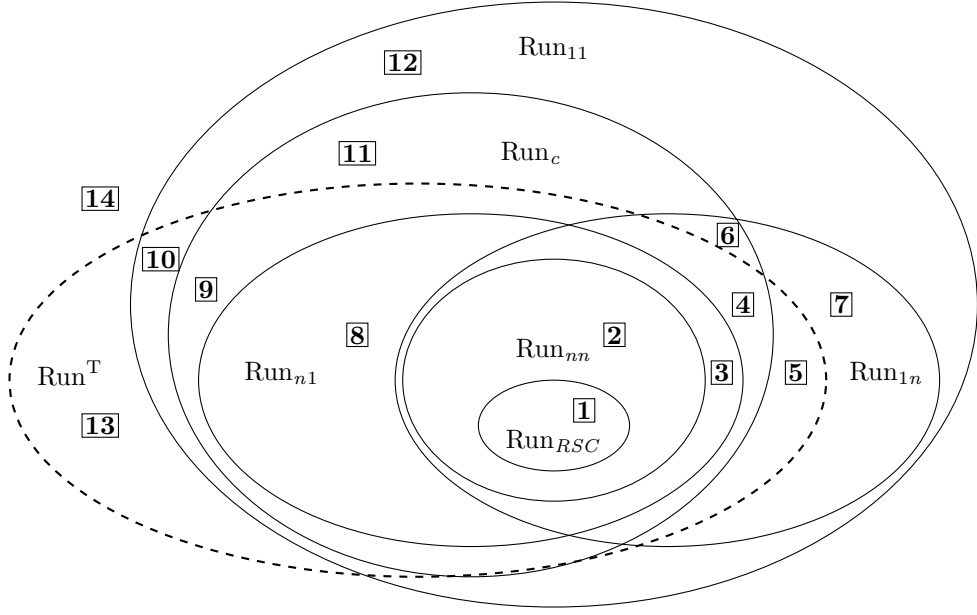


Figure 7.2: Inclusion of the Sets of Runs. Total ordering (dashed line) extends the existing hierarchy from Theorem 17 (page 39) and example runs prove strict inclusion. The examples associated to each label are presented in Table 7.1 and Table 7.2.

Run^A denotes the set of one-to-all runs that extend one-to-all distributed executions.

The second column in Table 7.1 and Table 7.2 presents one-to-all runs that may serve as examples for each region in the diagram in Figure 7.2. One cell in the table is missing: the example would correspond to a one-to-all run (Run^A) with total-ordering (Run^T) that is *FIFO 1-n* (Run_{1n}) but not *Causal* (Run_c). In the following theorem, we state and prove that there is no such run.

Theorem 44 (One-to-All Impossibility). *No one-to-all run can feature total-ordering and FIFO 1-n ordering without being also Causal.*

$$\text{Run}_{1n} \cap \text{Run}^T \cap \text{Run}^A \subsetneq \text{Run}_c$$

Proof. Let $\sigma \triangleq (E, (\leq_p)_{p \in \mathcal{P}}, \prec_c, \prec_\sigma, \text{com}, \text{peer}, \text{mes}) \in \text{Run}_{1n} \cap \text{Run}^T \cap \text{Run}^A$.

$$\text{Let } es_1, es_2, er_1, er_2 \in E \text{ such that: } \left(\begin{array}{l} \text{com}(es_1) = \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{com}(er_2) = \text{Receive} \\ \wedge \text{mes}(es_1) = \text{mes}(er_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) \\ \wedge \text{peer}(er_1) = \text{peer}(er_2) \\ \wedge es_1 \neq es_2 \\ \wedge es_1 \prec_c es_2 \end{array} \right)$$

Goal $er_1 \prec_c er_2$

By induction on the principle underlying Lemma 5 (page 24.

1. Base Case: $es_1 \prec_c^{\text{elem}} es_2$.

(a) First \prec_c^{elem} Case: $es_1 \prec_c es_2 \wedge \text{peer}(es_1) = \text{peer}(es_2)$.

QED because $\sigma \in \text{Run}_{1n}$.

(b) Second \prec_c^{elem} Case: $\text{com}(es_1) = \text{Send} \wedge \text{com}(es_2) = \text{Receive} \wedge \text{mes}(es_1) = \text{mes}(es_2)$.

Contradiction $\text{com}(es_2) = \text{Send}$.

2. Inductive Step: let $e \in E$ such that $es_1 \neq e \wedge es_1 \prec_c^{\text{elem}} e \wedge e \prec_c es_2$.

$$\text{By the induction hypothesis: } \left(\begin{array}{l} \text{com}(e) = \text{com}(es_2) = \text{Send} \\ \wedge \text{com}(er_1) = \text{com}(er_2) = \text{Receive} \\ \wedge \text{mes}(e) = \text{mes}(er_1) \\ \wedge \text{mes}(es_2) = \text{mes}(er_2) \\ \wedge \text{peer}(er_1) = \text{peer}(er_2) \\ \wedge e \neq es_2 \\ \wedge e \prec_c es_2 \end{array} \right) \Rightarrow e \prec_c er_2$$

(a) First \prec_c^{elem} Case: $es_1 \prec_c e \wedge \text{peer}(es_1) = \text{peer}(e)$.

i. Case $\text{com}(e) = \text{Send}$:

$\exists er \in E : \text{com}(er) = \text{Receive} \wedge \text{mes}(er) = \text{mes}(e) \wedge \text{peer}(er) = \text{peer}(er_1) = \text{peer}(er_2)$ because $\sigma \in \text{Run}^A$.

$er_1 \prec_c er$ since $es_1 \prec_c e \wedge es_1 \neq e \wedge \text{peer}(es_1) = \text{peer}(e)$ and $\sigma \in \text{Run}_{1n}$.

A. Case $e = es_2$:

QED because $\sigma \in \text{Run}_{1n}$.

B. Case $e \neq es_2$:

$er \prec_c er_2$ **by the induction hypothesis**.

QED by transitivity of \prec_c .

ii. Case $\text{com}(e) = \text{Internal} \vee \text{com}(e) = \text{Receive}$:

A. Case $\text{peer}(es_1) = \text{peer}(es_2)$:

QED because $\sigma \in \text{Run}_{1n}$.

B. Case $\text{peer}(es_1) \neq \text{peer}(es_2)$:

$$\exists es, er \in E : \left(\begin{array}{l} \text{com}(es) = \text{Send} \\ \wedge \text{com}(er) = \text{Receive} \\ \wedge \text{mes}(es) = \text{mes}(er) \\ \wedge \text{peer}(es) = \text{peer}(er_1) \\ \wedge e \prec_c es \\ \wedge er \prec_c er_2 \end{array} \right) \text{ by Theorem 6 (page 24) since}$$

$\text{peer}(e) = \text{peer}(es_1) \neq \text{peer}(es_2)$ and $e \prec_c er_2$.

$es_1 \prec_c es$ by transitivity because $es_1 \prec_c e$ and $e \prec_c es$.

$\text{peer}(es_1) = \text{peer}(e) = \text{peer}(es)$.

$e \prec_c er_2$ by transitivity because $es \prec_c er$ (definition of a distributed execution) and $er \prec_c er_2$.

Proof by case 2.a.i. with $e \leftarrow es$.

(b) Second \prec_c^{elem} Case: $\text{com}(es_1) = \text{Send} \wedge \text{com}(e) = \text{Receive} \wedge \text{mes}(es_1) = \text{mes}(e)$.

$\exists er'_2 \in E : \text{com}(er'_2) = \text{Receive} \wedge \text{mes}(es_2) = \text{mes}(er'_2) \wedge \text{peer}(er'_2) = \text{peer}(e)$ because $\sigma \in \text{Run}^A$.

$es_2 \prec_c er'_2$ by definition of a distributed execution.

$e \prec_c er'_2$ by transitivity of \prec_c because $e \prec_c es_2$ and $es_2 \prec_c er'_2$.

QED $er_1 \prec_c er_2$ because $e \prec_c er'_2$ and $\sigma \in \text{Run}^T$ (the order of the receptions must be the same on all the peers).

In Table 7.1 and Table 7.2, many examples such as #8 prove that $\text{Run}_c \not\subseteq \text{Run}_{1n} \cap \text{Run}^T \cap \text{Run}^A$. \square

7.2 Towards a Framework for the Compatibility Checking of Compositions with Asynchronous Multicast Communication

This section deals with the design of a verification framework in TLA^+ that retains the features of the mechanisation of the point-to-point framework described in Chapter 5 and also handles multicast communication. The following is a snapshot of ongoing work. After motivating the choices made to address the challenges of multicast communication, we present the current state of the multicast verification framework. Eventually, we discuss the limits and issues that are still to be addressed.

7.2.1 Lifespan of a Message in the Network of Messages In-Transit

In the mechanisation of the point-to-point framework for the compatibility checking of compositions of peers, the transmission of a message consists of two operations:

1. Send. The message is added to a set of messages in transit called the network.
2. Receive. If the message to be received is in transit in the network, it is removed from the network.

This means that once a message has been received, it cannot be received again later because it is no longer in the network of in transit messages. This constitutes the first and main challenge to be addressed when it comes to multicast communication.

Sending the messages over and over

A simple solution would consist in sending the message again once it has been received so it can be received another time by another peer. There are two problems:

- The reception of a message may be forbidden by the communication model and the ordering property depends on the order of entry in the network. For instance, with *FIFO* ordering and two messages m_1 and m_2 sent in this order, the communication model should guarantee that m_1 is received before m_2 . If m_1 is received and then sent again, the new order of entry in the network becomes m_2 followed by m_1 .
- This solution does not specify when to stop sending messages again, that is to say the lifespan of a message in the network.

Never removing the messages from the network

If we were to leave messages in the network of messages in transit forever, this would indeed allow multicast communication : a message could be received as many times as the peers want. However, this poses a problem with some communication models such as *FIFO n-n*. In the operational description of *FIFO n-n* in the point-to-point framework, it is not possible to ignore a message in the network contrary to models such as *FIFO 1-1* in which channels may not be “listened to” anymore. This means that the first message to enter the network would block the reception of all the others.

Removing a message from the network once received by all the peers

A solution to the previous issue consists in removing a message from the network once all the peers have received it. This means that all the peers must be ready to receive all the messages in order not to block the system. This requirement is too strong to allow for the verification of interesting and realistic systems.

Removing a message from the network after the relevant peers have received it

Removing a message from the network as soon as the peers that should receive it actually have solves the previous problem. This is reminiscent of Chapter 4 when a solution was devised to prevent a peer from having to be ready to receive messages transmitted between other independent peers; in other words the specification of a peer should not depend on the environment it takes part in. Chapter 4 develops the concepts of listened channels and stability with regard to interest to overcome the problem.

With a similar notion of interest, removing a message from the network once it has been received by all the interested peers seems like the most sensible management of the lifespan of a message in transit. It also sticks to the guidelines of the point-to-point framework which paves the way for compatibility with point-to-point communication.

7.2.2 Preventing a Peer from Receiving the Same Message Twice

According to Definition 8 (page 26) of multicast communication in Chapter 2, a message should be received at most once per peer.

- A message might remain in the network after it has been received once. The state of the network cannot prevent duplicate receptions on a peer.
- Interest deals with channels, not messages. If a peer stops being interested in a channel, it cannot receive new messages from this channel later which is not an acceptable restriction.

Therefore, a new mechanism has to be introduced in order to prevent local duplicate receptions. It simply consists of local sets *received* of already received messages. Once a message is received, on peer p , it is added to $received[p]$ and a message m can only be received on p if $m \notin received[p]$.

7.2.3 Specification of the Interest

In the framework in Chapter 4, the interest, which is part of the specification of the peers, is a label on the reception: the “listened channels”. It contains the set of channels the peer is still interested in at the time of the reception. The communication model may make use of this information to allow or forbid the reception.

Here, in order to ease the management of messages in the network, that is to say removing them when no peer is interested anymore, we remove this label and introduce a new action in the communication models called *ignore*. The specifications of the peers can make use of *ignore* to state that they are no longer interested in a given set of channels. The communication model itself keeps track of which peer is interested in which channel with a state variable *interest* that maps each peer to a set of channels. The specification of the *ignore* action may consist in removing a message from the network when the last peer that is interested in a channel stops to be. Specifying interest with the *ignore* action also forbids, by construction, specifications that are not stable with regard to interest. It is indeed impossible to stop ignoring a channel.

Initially, a peer may be interested in some channels or all of them. During the course of the run, a peer can lose interest in some of those channels thanks to the *ignore* action which prevents the communication model from imposing the reception of irrelevant messages but the peer cannot cheat and resume interest in these channels: it is unable to violate stability with regard to interest.

receive If the reception is possible on peer p (the message is not in $received[p]$, the channel is in $interest[p]$, and the ordering policy of the communication model is respected), the message is added to $received[p]$. It is removed from the network only if this was the last interested peer that had not yet received the message.

ignore The set of channels is removed from the channels of interest of peer p , that is to say from $interest[p]$. If there were messages in transit on channels that no longer interest any peer, they are removed from the network.

Alternatively, explicit group destination could have been used instead, like the explicit destination peers in the communication models in Chapter 6. A message, at send time, would be associated to a set of destination peers and removed from the network once all these peers have received the message. Basing the lifespan of a message on the concept of interest, much like in the framework described in Chapter 4 and Chapter 5, allows to specify systems where channels are not statically associated to a given sender or a given group of receivers.

7.2.4 Point-to-Point and One-to-All Communication

This revision of the framework for multicast communication aims at handling systems with point-to-point communication as well, or even systems that mix multicast and point-to-point communication thanks to composite communication models as in the point-to-point framework of Chapter 5.

The operational semantics of multicast communication described previously is adapted to become generic and also describe point-to-point communication. Consider two parameters of the communication called *MIN* and *MAX*. Let N denote the number of peers in the system.

- *MIN* is the minimum number of times a message must be received before it is removed from the network of messages in transit.
- *MAX* is the maximum number of times a message can be received before it is removed from the network, even if there still are peers that are interested in the message.

Up until now, we have described *multicast 0-N* communication: a message is removed from the network when all the peers that are interested in this message have received it. Point-to-point communication corresponds to *multicast 1-1*. Indeed, a message must be received at least once before it can be removed from the network and must not be received more than once. This means it is immediately removed from the network following the first reception, never before. Similarly, *multicast 1-n* would correspond to multicast communication where at least one peer must receive a message before it is removed, and *multicast n-n* only generates one-to-all runs.

7.2.5 Organisation and Structure of the TLA⁺ Modules

In this framework, we split the ordering properties and the management of the network into different TLA⁺ modules to avoid code duplication and easy interfacing of various properties on the communication. For example regular multicast *FIFO 1-1* bound to 5 messages in transit

is the composite communication model composed of: the module for *multicast* communication (with $MIN = 0$ and $MAX = N$), the module for *FIFO 1-1* ordering, and the module for bounded communication (with a value of 5 for the bound). Only the *multicast* module removes messages from its network. When this happens, the networks of the other communication models have to be updated accordingly which requires an additional transition. A new boolean state variable *ready* prevents communication actions from occurring before the update is done and the states of the communication models are consistent.

In Chapter 5, Section 5.5.1 details alternative specifications of the communication models where the histories do not contain messages but message identifiers instead. Thus, the histories and network are flat sets of identifiers. This is a simplification over the recursive data structure of the message histories of the point-to-point framework. The multicast framework relies on identifiers and given a message, its identifier is the same across all the modules of a composite communication model.

Minor small details, everything else remains identical to the point-to-point framework from Chapter 5. The different TLA⁺ modules are:

peermanagement Defines the array of state variables (program counters) *peers* for the peers and provides helper actions and basic properties. See Figure 7.4. Almost identical to its counterpart from Chapter 5.

system Specifies the composition of peers and instantiates the composite communication model to use. Contrary to the framework from Chapter 5 and as explained earlier, the *ignore* action of the communication model is used instead of listened channels that parameterise the *receive* action. Figure 7.3 is a simple example that highlights the small differences (*ignore* action) with specifications of compositions in the framework from Chapter 5. It may be specified manually or derived from a transition system or a CCS term as in Chapter 5.

multicom The composite communication model. The idea of a conjunction between the communication predicates of the submodels remains the same as in Chapter 5 but this module exhibits many particularities (detailed later) of the multicast framework. This module is generated according to the number of models and the channels they are associated to.

multicast The module that, depending on the *MIN* and *MAX* value, specifies whether or not a message should be removed from the network.

Communication Models A module per communication model that specifies the underlying ordering policy. The seven communication models have been specified. Ongoing work aims at specifying a communication model that models total ordering in multicast communication.

7.2.6 Communication Models

In order to ease interfacing of the modules, the communication models have a single state variable *s*, a TLA⁺ record. The fields correspond for instance to the network or the interest. Besides the *ignore* action, another new action *free* is present in the specifications. It is used by the *multicom* during the update of the networks of the communication models and consists in removing messages in a given set of identifiers from the network and message histories. Furthermore, the specifications provide the *used_id* set that contains all the identifiers that are not free to be affected to new messages. The *multicom* specification makes use of the information for the generation of new message identifiers.

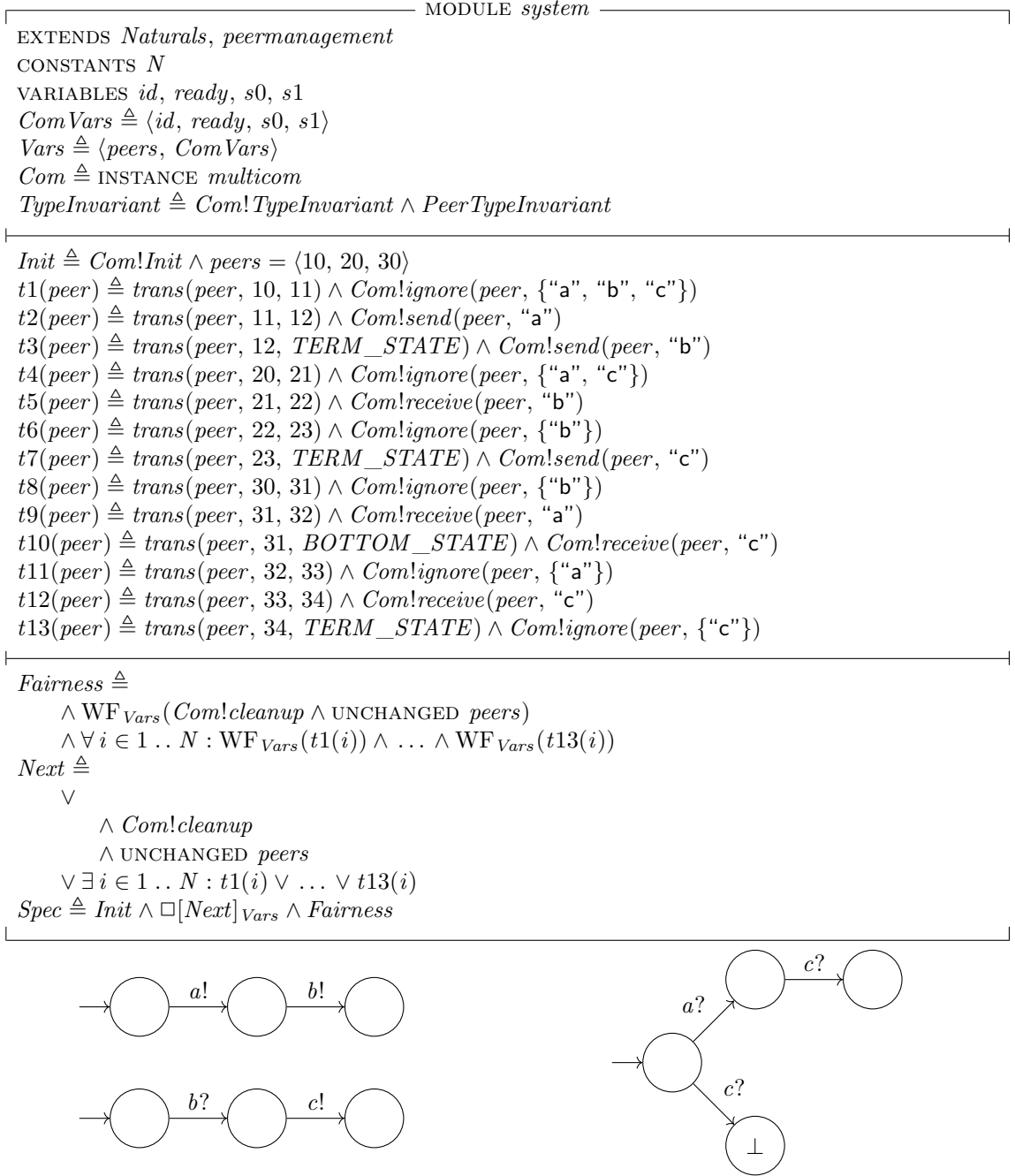


Figure 7.3: TLA⁺ Module of the Specification of a Composition. It corresponds to the classic example that illustrates causality with three peers and three messages. It is a point-to-point example but here illustrate how the *ignore* actions are used.

MODULE *peermanagement*

EXTENDS *Naturals, Sequences*

CONSTANT

BOTTOM_STATE,
TERM_STATE

VARIABLE

peers

PeerTypeInvariant \triangleq *peers* \in *Seq*(*Nat*)

trans(*peer*, *init*, *next*) \triangleq

\wedge *peers*[*peer*] = *init*
 \wedge *peers'* = [*peers* EXCEPT ![*peer*] = *next*]

AllPeersIn(*states*) \triangleq $\forall i \in \text{DOMAIN } \textit{peers} : \textit{peers}[i] \in \textit{states}$

OnePeerIn(*states*) \triangleq $\exists i \in \text{DOMAIN } \textit{peers} : \textit{peers}[i] \in \textit{states}$

NonBottom \triangleq $\Box \neg \textit{OnePeerIn}(\{\textit{BOTTOM_STATE}\})$

Terminates \triangleq $\Diamond \Box \textit{AllPeersIn}(\{\textit{TERM_STATE}\})$

Terminated \triangleq *AllPeersIn*($\{\textit{TERM_STATE}\}$)

Figure 7.4: TLA⁺ Module that eases the management of the peers in a composition. It provides the *trans* action to change the state of a peer and defines temporal properties that might serve as a basis for compatibility checking.

A Regular Communication Model

The specifications of the seven communication models have not changed. As explained before, the TLA⁺ modules only specify the ordering policy. The following is the specification of the *FIFO 1-1* communication model.

MODULE *fifo11*

EXTENDS *Naturals*

CONSTANTS

N, Number of peers
CHANNEL Set of channels

VARIABLES *s*

Peer \triangleq 1 .. *N*

MessageHistory \triangleq SUBSET *Nat*

Message \triangleq

[

id : *Nat*,
channel : *CHANNEL*,
peer : *Peer*,
history : *MessageHistory*

]

Network \triangleq SUBSET *Message*

Interest \triangleq [*Peer* \rightarrow SUBSET *CHANNEL*]

TypeInvariant \triangleq

s \in [

$$\begin{array}{l}
\text{network} : \text{Network}, \\
\text{interest} : \text{Interest} \\
] \\
\text{Init} \triangleq \\
s = [\\
\text{network} \mapsto \{\}, \quad \text{Empty network} \\
\text{interest} \mapsto [\text{peer} \in \text{Peer} \mapsto \text{CHANNEL}] \quad \text{Full interest} \\
] \\
\text{TransitingMessages} \triangleq s.\text{network} \neq \{\}
\end{array}$$

$$\begin{array}{l}
\text{IGNORE} \\
\text{peer} \quad \text{peer losing interest in the channels} \\
\text{chan_set} \quad \text{set of channels to ignore} \\
\text{ignore}(\text{peer}, \text{chan_set}) \triangleq \\
\wedge s' = [s \text{ EXCEPT} \\
\quad !.\text{interest} = [s.\text{interest} \text{ EXCEPT} \\
\quad \quad ![peer] = s.\text{interest}[peer] \setminus \text{chan_set} \\
\quad] \\
] \\
\text{SEND} \\
\text{id} \quad \text{id to give to the new message} \\
\text{peer} \quad \text{peer that sends the new message} \\
\text{chan} \quad \text{channel associated to the message} \\
\text{send}(\text{id}, \text{peer}, \text{channel}) \triangleq \\
\text{LET } \text{related_messages} \triangleq \{m \in s.\text{network} : m.\text{peer} = \text{peer}\} \text{ IN} \\
s' = [s \text{ EXCEPT} \\
\quad !.\text{network} = @ \cup \\
\quad \{ \\
\quad \quad [\\
\quad \quad \quad \text{id} \mapsto \text{id}, \\
\quad \quad \quad \text{channel} \mapsto \text{channel}, \\
\quad \quad \quad \text{peer} \mapsto \text{peer}, \\
\quad \quad \quad \text{history} \mapsto \text{UNION } \{m.\text{history} \cup \{m.\text{id}\} : m \in \text{related_messages}\} \\
\quad \quad] \\
\quad \} \\
] \\
\text{RECEIVE} \\
\text{id} \quad \text{id the received message must match} \\
\text{peer} \quad \text{peer that receives the message} \\
\text{chan} \quad \text{channel associated to the message} \\
\text{receive}(\text{id}, \text{peer}, \text{channel}) \triangleq \\
\exists m \in s.\text{network} : \\
\quad \wedge m.\text{id} = \text{id} \\
\quad \wedge m.\text{channel} = \text{channel} \\
\quad \wedge \neg \exists m2 \in s.\text{network} : \quad \text{Ordering policy}
\end{array}$$

$$\begin{aligned}
& \wedge m2.channel \in s.interest[peer] \\
& \wedge m.peer = m2.peer \\
& \wedge m2.id \in m.history \\
& \wedge \text{UNCHANGED } s \quad \text{Not a transition: it only specifies the ordering policy}
\end{aligned}$$

FREE

ids set of *ids* to remove from the network

$$\begin{aligned}
free(ids) &\triangleq \\
s' &= \\
&[s \text{ EXCEPT} \\
&\quad !.network = \\
&\quad \{ \\
&\quad \quad [m \text{ EXCEPT } !.history = @ \setminus ids] : \\
&\quad \quad m \in \{m2 \in @ : m2.id \notin ids\} \\
&\quad \} \\
&]
\end{aligned}$$

Message *ids* currently in use in this model

$$\begin{aligned}
used_ids &\triangleq \{m.id : m \in s.network\} \\
transiting_ids &\triangleq \{m.id : m \in s.network\}
\end{aligned}$$

The Multicast Communication Model

The rules that specify the *multicast* module have already been detailed thoroughly. It is the only communication model that removes messages from its network by itself depending on the parameters *MIN* and *MAX*.

MODULE *multicast*

EXTENDS *Naturals*

CONSTANTS

<i>N</i> ,	Number of peers
<i>CHANNEL</i> ,	Set of channels
<i>MIN</i> ,	After <i>MIN</i> receptions, a message can be removed by disinterest
<i>MAX</i>	After <i>MAX</i> receptions, a message is removed no matter the interest

VARIABLES *s*

Peer $\triangleq 1 \dots N$

MessageHistory $\triangleq \text{SUBSET } Nat$

Message \triangleq

[

id : *Nat*,

channel : *CHANNEL*,

received : *Nat* The number of times a message has been received

]

There is no message history because this model does not deal with message ordering

Network $\triangleq \text{SUBSET } Message$

Interest $\triangleq [Peer \rightarrow \text{SUBSET } CHANNEL]$

Received $\triangleq [Peer \rightarrow \text{SUBSET } Nat]$

TypeInvariant \triangleq

```

    s ∈ [
      network : Network,
      interest : Interest,
      received : Received
    ]
  Init ≜
    s = [
      network ↦ {},
      interest ↦ [peer ∈ Peer ↦ CHANNEL],
      received ↦ [peer ∈ Peer ↦ {}]
    ]
  TransitingMessages ≜ s.network ≠ {}

```

```

  IGNORE
  peer      peer losing interest in the channels
  chan_set  set of channels to ignore
  ignore(peer, chan_set) ≜
    LET new_peer_interest ≜ s.interest[peer] \ chan_set IN
      s' = [s EXCEPT
        !.interest =
          [ @ EXCEPT
            ![peer] = new_peer_interest
          ],
        !.network =
          { m ∈ @ :
            ∨ m.received < MIN
            ∨ ∃ p ∈ Peer \ {peer} : m.channel ∈ s.interest[p]
            ∨ m.channel ∈ new_peer_interest
          }
      ]

```

```

  SEND
  id        id to give to the new message
  peer      peer that sends the new message
  chan      channel associated to the message
  send(id, peer, channel) ≜
    LET related_messages ≜ { m ∈ s.network : m.peer = peer } IN
      s' = [s EXCEPT
        !.network = @ ∪
          {
            [
              id ↦ id,
              channel ↦ channel,
              received ↦ 0
            ]
          }
      ]

```

```

  RECEIVE

```

id id the received message must match
 $peer$ peer that receives the message
 $chan$ channel associated to the message
 $receive(id, peer, channel) \triangleq$
 $\exists m \in s.network :$
 $\wedge m.id = id$
 $\wedge m.channel = channel$
 $\wedge m.id \notin s.received[peer]$
 $\wedge LET network_preupdate \triangleq$ The incremented values of the receptions counters are computed
 $(s.network \setminus \{m\}) \cup \{[m \text{ EXCEPT } !.received = @ + 1]\}$ IN
 $s' = [s \text{ EXCEPT}$ The network is updated accordingly
 $! .network = \{m2 \in network_preupdate :$ Receiving might remove a message.
 $\wedge m2.received < MAX$ Between MIN and MAX ,
 \wedge messages are kept if
 $\vee m2.received < MIN$ their channel interests
 $\vee \exists p \in Peer : m2.channel \in s.interest[p]$ a peer.
 $\},$
 $! .received = [@ \text{ EXCEPT } ![peer] = @ \cup \{id\}]$
 $]$

FREE
 ids set of ids to remove from the network
 $free(ids) \triangleq$
 $s' =$
 $[s \text{ EXCEPT}$
 $! .network = \{m \in @ : m.id \notin ids\}$
 $]$

 Message ids currently in use in this model
 $used_ids \triangleq \{m.id : m \in s.network\}$
 $transiting_ids \triangleq \{m.id : m \in s.network\}$

7.2.7 A Composite Communication Model

After a communication action, the state of the composite communication model becomes $\neg ready$. The *cleanup* action can switch it back to *ready* after it has computed which identifiers correspond to messages that should be removed from the networks of the communication models that take part in *multicom*. Those identifiers are those “in use” that are not “in transit” where *in_use* is the union of all the identifiers used in the submodels and *in_transit* the intersection. Since messages are not removed from the communication models that specify ordering policies, their identifiers are in *in_use* but since the messages may be removed from the network of the *multicast* model, their identifiers are not in *in_transit*. This is how the removal of messages from the network propagates through all the communication models and how their state is kept consistent.

MODULE *multicom*
 EXTENDS *Naturals*
 CONSTANTS N Number of peers
 VARIABLES $id, ready, s0, s1$
 Vars $\triangleq \langle id, ready, s0, s1 \rangle$

Groups of channels

$Channel0 \triangleq \{\text{"a"}, \text{"b"}, \text{"c"}\}$
 $Channel1 \triangleq \{\text{"a"}, \text{"b"}, \text{"c"}\}$
 $Channel \triangleq Channel0 \cup Channel1$
 $Com0 \triangleq \text{INSTANCE } multicast \text{ WITH } CHANNEL \leftarrow Channel0, MIN \leftarrow 0, MAX \leftarrow N, s \leftarrow s0$
 $Com1 \triangleq \text{INSTANCE } fifo11 \text{ WITH } CHANNEL \leftarrow Channel1, s \leftarrow s1$

$IdTypeInvariant \triangleq id \in [$
 $\quad in_use \quad : \text{SUBSET } Nat,$
 $\quad in_transit \quad : \text{SUBSET } Nat$
 $\quad] \wedge id.in_transit \subseteq id.in_use$
 $TypeInvariant \triangleq$
 $\quad \wedge Com0!TypeInvariant \wedge Com1!TypeInvariant$
 $\quad \wedge IdTypeInvariant$
 $\quad \wedge ready \in \text{BOOLEAN}$

$Init \triangleq$
 $\quad \wedge Com0!Init \wedge Com1!Init$
 $\quad \wedge id = [$
 $\quad \quad in_use \quad \mapsto \{\},$
 $\quad \quad in_transit \quad \mapsto \{\}$
 $\quad]$

$TransitingMessages \triangleq Com0!TransitingMessages \wedge Com1!TransitingMessages$

$max(id_set) \triangleq$
 $\quad \text{IF } id_set = \{\}$
 $\quad \quad \text{THEN } 0$
 $\quad \quad \text{ELSE } (\text{CHOOSE } x \in id_set : (\forall y \in id_set : y \leq x))$
 $first_free(id_set, used) \triangleq$
 $\quad \text{CHOOSE } x \in id_set \setminus used : (\forall y \in id_set \setminus used : y \geq x)$

SEND
 $peer$ peer that sends the new message
 $chan$ channel associated to the message
 $send(peer, chan) \triangleq$
 $\quad \text{LET } new_id \triangleq first_free(1 \dots max(id.in_use) + 1, id.in_use) \text{ IN}$
 $\quad \quad \wedge ready$
 $\quad \quad \wedge ready' = \text{FALSE}$
 $\quad \quad \wedge$
 $\quad \quad \vee$
 $\quad \quad \quad \wedge chan \in Channel0$
 $\quad \quad \quad \wedge chan \notin Channel1$
 $\quad \quad \quad \wedge Com0!send(new_id, peer, chan)$
 $\quad \quad \quad \wedge \text{UNCHANGED } s1$
 $\quad \quad \vee$
 $\quad \quad \quad \wedge chan \notin Channel0$
 $\quad \quad \quad \wedge chan \in Channel1$
 $\quad \quad \quad \wedge \text{UNCHANGED } s0$

$$\begin{aligned}
& \wedge Com1!send(new_id, peer, chan) \\
\vee \\
& \wedge chan \in Channel0 \\
& \wedge chan \in Channel1 \\
& \wedge Com0!send(new_id, peer, chan) \\
& \wedge Com1!send(new_id, peer, chan) \\
\wedge id' = & \\
& [\\
& \quad in_use \mapsto \text{UNION } \{Com0!used_ids', Com1!used_ids'\}, \\
& \quad in_transit \mapsto Com0!transiting_ids' \cap Com1!transiting_ids' \\
&]
\end{aligned}$$

RECEIVE

peer peer that receives the message

chan channel associated to the message

$receive(peer, chan) \triangleq$

$\exists common_id \in id.in_transit :$

$\wedge ready$

$\wedge ready' = \text{FALSE}$

\wedge

\vee

$\wedge chan \in Channel0$

$\wedge chan \notin Channel1$

$\wedge Com0!receive(common_id, peer, chan)$

$\wedge \text{UNCHANGED } s1$

\vee

$\wedge chan \notin Channel0$

$\wedge chan \in Channel1$

$\wedge \text{UNCHANGED } s0$

$\wedge Com1!receive(common_id, peer, chan)$

\vee

$\wedge chan \in Channel0$

$\wedge chan \in Channel1$

$\wedge Com0!receive(common_id, peer, chan)$

$\wedge Com1!receive(common_id, peer, chan)$

$\wedge id' =$

$[$

$in_use \mapsto \text{UNION } \{Com0!used_ids', Com1!used_ids'\},$

$in_transit \mapsto Com0!transiting_ids' \cap Com1!transiting_ids'$

$]$

IGNORE

peer peer losing interest in the channels

chan_set set of channels to ignore

$ignore(peer, chan_set) \triangleq$

$\wedge ready$

$\wedge ready' = \text{FALSE}$

$\wedge Com0!ignore(peer, chan_set) \wedge Com1!ignore(peer, chan_set)$

$\wedge id' =$

$[$

```

    in_use    ↦ UNION { Com0!used_ids', Com1!used_ids' },
    in_transit ↦ Com0!transiting_ids' ∩ Com1!transiting_ids'
  ]
]

CLEANUP
Free ids that no longer belong to messages in transit
cleanup ≜
  ∧ ¬ready
  ∧ ready' = TRUE
  ∧ LET useless_ids ≜ id.in_use \ id.in_transit IN
    ∧ Com0!free(useless_ids) ∧ Com1!free(useless_ids)
    ∧ id' =
      [
        in_use    ↦ UNION { Com0!used_ids', Com1!used_ids' },
        in_transit ↦ Com0!transiting_ids' ∩ Com1!transiting_ids'
      ]

```

7.2.8 Limitations

In the current state of the framework, it is impossible to check the compatibility of a system under a composite communication model that has different groups of channels. In the previous example of a *multicom* module, there were two identical groups associated respectively to a *multicast* instance (with $MIN = 0$ and $MAX = N$) and a *fifo11* instance, hence modelling *multicast 0-n FIFO 1-1* communication. This kind of configuration works and the *cleanup* action ensures the consistence of the networks in the two instances.

With a composite communication model such as the one used in the examination management system example (Section 5.4.1 in Chapter 5 on page 100), the *cleanup* action is problematic. Let's consider 4 instances in a composite communication model:

1. *Multicast 0-n* on channels a, b, c .
2. *FIFO 1-1* on channels a, b, c .
3. *Multicast 1-1* on channels d, e .
4. *FIFO n-n* on channels d, e .

The *cleanup* action removes messages that are not common to all the four networks. This is fine with only 1 and 2, or only 3 and 4, because the groups of channels are the same. However, with the four instances, there is no way a message on channel a will ever end up in the networks of instances 3 or 4. Conversely, no message on channels d can be in the networks of instances 1 and 2. This means after each communication action, the *cleanup action* removes all the messages from all the networks. If some groups overlapped, only a few common messages would survive.

When a message is removed from the network of a *multicast* instance but remains in another instance of a communication model (for example *fifo11*), there is no risk to receive the message because the *receive* action in a *multicom* module is a conjunction of all the *receive* actions of the instantiated communication models, including *multicast* where the *receive* action would be disabled. Therefore, the *cleanup* operation might seem unnecessary. It actually is not for two reasons.

- The *cleanup* operation prevents the network from indefinitely accumulating messages. In systems that loop over a finite number of state, the state-space during model checking would diverge anyway. This is reminiscent of the problem described in Section 5.5.1 of Chapter 5 with message histories and purge of received messages.
- It is possible to ignore received messages as if they were not in the network anymore in communication models such as *FIFO 1-1*, *FIFO n-1*, or *Causal* thanks to the ordering property that takes interest into account. In the *FIFO 1-n*, *FIFO n-n*, or *RSC* communication models, it is however impossible to ignore messages in the network. Take *FIFO n-n* as an example with two messages m_1 and m_2 added to the network in this order. After receiving m_1 in point-to-point communication (*Multicast 1-1*), m_1 is removed from the network of the *multicast* instance which prevents it from being received again but it is still in the network of the *fifonn* instance which will block any further reception.

As a conclusion, the *cleanup* operation is almost always necessary. The following table sums the capabilities of both versions of the framework, with, or without a *cleanup* operation.

	With <i>cleanup</i>	Without <i>cleanup</i>
Homogeneous groups of channels	Yes	Yes
Heterogeneous groups of channels	Invalid	Yes
Systems with loops	Maybe finite state-space	Infinite state-space
<i>FIFO 1-n</i> , <i>FIFO n-n</i> , and <i>RSC</i>	Yes	Invalid

7.2.9 Addressing the Issue

Several ideas may address the issue:

Individual Network Management In the point-to-point framework in Chapter 5, each communication model manages additions and removals of messages in their network. If the multicast specification were duplicated in every communication model, there would be no need for the *multicast* module and the previous issue would not arise. The specifications of the communication models would grow in complexity with a significant degree of duplication since the *multicast* module is quite complex.

Groups of network consistence The specification of a composite communication model could group instances of communication models into groups of network consistence, i.e. groups of instances that should keep their network identical. For instance, the layout from 7.2.8 would be split into two of these higher level groups:

- First group of instances
 - *Multicast 0-n* on channels a, b, c .
 - *FIFO 1-1* on channels a, b, c .
- Second group of instances
 - *Multicast 1-1* on channels d, e .
 - *FIFO n-n* on channels d, e .

The *cleanup* action would then operate individually on each group of instances. The composite communication models in the point-to-point framework allow to specify groups of channels that overlap and this solution would also allow it inside a given group of instances. In the following example, the *Causal* ordering policy applies to messages on channels a, b ,

and c , and *FIFO 1-1* applies to messages on channels c and d . There is, for instance, no constraint on the order of the receptions between messages on a and d . Channels e and f are part of another communication group that is point-to-point: the network management is different.

- First group of instances
 - *Multicast 0-n* on channels a, b, c, d .
 - *Causal* on channels a, b, c .
 - *FIFO 1-1* on channels c, d .
- Second group of instances
 - *Multicast 1-1* on channels e, f .
 - *FIFO n-n* on channels e, f .

Ongoing work aims at exploring these ideas.

7.3 Conclusion

This chapter has introduced total ordering in multicast communication to our existing formal toolkit, along with one-to-all communication, and proved how they integrate to the hierarchy of communication models. The results illustrate the relative complexity of multicast communication in comparison with point-to-point communication. Second, we have proposed a revision of the TLA⁺ framework for compatibility checking that handles both multicast communication and point-to-point communication as a special case of multicast communication. The overall structure remain similar to the existing framework minor small choices that result in a greater distance between the formal base and the mechanisation. The multicast framework indeed lacks the proofs of correction and completeness that give trust to its point-to-point counterpart. Along with some fine-tuning on the specification of the lifespan of a message in composite communication model and adaptation of the user-friendly automations of the point-to-point framework, proving the multicast framework conforms to the definitions of the communication models is a prime goal.

Part IV

Conclusion

Chapter 8

Conclusion and Future Work

The compatibility of communicating peers is seldom trivial. In particular, with asynchronous communication, the decoupling of the emissions and deliveries of messages allow for interleaving of communication events, chaotic ordering of reception, and increased risks of incompatibility. This work contributes to the design and verification of safe and trustworthy distributed systems by providing comprehensive formal tools that take the diversity of asynchronous interactions into account. The first section sums up this work and assesses the fulfillment of the initial goals. Then, the second section details areas of improvement that are to be tackled in future work.

8.1 Results and Practical Benefits

This work addresses classic issues in the formal approach of asynchronous communication:

- the lack of clarity in the denomination and distinction between common communication models;
- the lack of consistency and uniformity in the specifications of these models;
- the lack of a concise big picture that highlights how all these interaction models relate to each other.

We answer these problems on two levels:

Logical Chapter 2 formally defines the communication models. The definitions all rely on distributed executions and runs, that is to say partially ordered sets of communication events and their linear extensions. They consist of an ordering property that give meaning to each model. The chapter also compares these definition to establish a hierarchy. Chapter 7 then extends this hierarchy to multicast-specific concepts.

Operational Chapter 4 provides operational specification of the same models and prove the conformance with the structural definitions. Chapter 6 proves the hierarchy holds with the operational descriptions and draws a big picture that encompasses the seven considered communication models and different degrees of abstractness.

Thanks to this work on the formalisation of asynchronous interactions, we provide a framework for the verification of safety and liveness properties in practical distributed systems that relies on a comprehensive formalisation of asynchronous interactions. The said framework is fully

mechanised, user-friendly, and proved conform to the theory. It has already been successfully used on a lot of examples. In Section 8.2.2, we describe how it helped make a hypothesis about the communication models and compatibility properties that constitute a key perspective.

Moreover, the “menagerie of refinements” established in Chapter 6 provides a useful map to direct the choice or substitution of a communication model depending on the desired communication properties and degree of abstractness.

Eventually, all the point-to-point exclusive contributions are to be extended to encompass group communication. Chapter 7 already presents a significant part of this work, including an early stage mechanised framework for the verification of distributed systems with group communication.

8.2 Future Work

8.2.1 Wider Range of Communication Models

The asynchronous interactions are not limited to the seven communication models we have studied in depth throughout the previous chapters. Future work aims at expanding the results to all kinds of new communication models.

Concrete Specifications

Chapter 6 considers a few concrete specifications of the models that are closer to actual implementations. This could be generalised to all the models with additional steps of refinement that lead to the said implementations. The main challenge is localisation, that is, associating each variable of the system to a peer. Some communication models, like *FIFO n-n*, are indeed more easily described in a centralised fashion that is not fit for implementation in distributed systems.

Capped Communication Models

Neither the structural definitions nor the certified operational specifications of the communication models actually allow for the specification of bounds on the number of messages in transit, globally, per channel, or per buffer. Although the framework for compatibility checking already makes it possible to specify a cap on the number of messages in transit in the network, it would also be interesting to compare the models and establish conditional hierarchies that depend on parameters such as the size and nature (global, local) of the bounds.

Group Communication

The structural definitions of the seven communication model are generic but only the point-to-point operational specifications of these models are proved to conform to these definitions. The proofs of correctness and completeness for the specifications of the seven communication models in the multicast framework 7 are to be carried out in future work since they require adaptation and tweaking. Future work also include the specification of a communication model for total ordering in the multicast framework. Currently, the menagerie of refinements does not include specifications for multicast communication. This would require to formalise the concept of destination group and specifications for the total ordering model.

Fault Models

Another perspective is the specification and comparison of fault models that allow to specify faults in distributed systems such as message loss, duplication, or crash of a peer. The specification of message loss is for instance a challenge that requires to decide on several questions. The concept may indeed refer to messages that are forever in transit in the network or to a removal from the network without an associated reception. Moreover, the specifications of the ordering policies should state whether or not the loss of a message allows the reception of newer messages, as if the lost message had never existed.

Round-Based Communication Models

A class of distributed algorithms models communication with rounds: at a given round, all the messages of the previous round are received and new messages that will be received during the next round are sent. Although reminiscent of synchronous communication (all the messages of the previous round are received at the same logical time), there is a decoupling of the emission and delivery of a message so the communication is asynchronous. In the different hierarchies, this new communication model would likely settle between the *RSC* communication model and the *FIFO n-n* communication model.

8.2.2 Equivalence between Communication Models with regard to Compatibility

This work has established and proved how the communication models and their alternate specifications relate to each other. Chapter 4 proves the specifications of the communication models with message histories conform to the structural definitions in Chapter 2. Chapter 2 compares these definitions and establishes a hierarchy that encompasses multicast communication while Chapter 7 expands the results with additional knowledge that is specific to multicast communication. Chapter 6 exhibits a “menagerie of refinements” for the point-to-point communication models that unifies results from the other chapters and also considers concrete specifications that are close to actual implementations. Thanks to these results, it is possible to know whether or not a communication model can be substituted by another without risking unexpected behaviour of the underlying distributed system.

However, these results only deal with the communication models by themselves, not the distributed systems they take part in or the considered compatibility properties. Interestingly, observations lead on many different examples with the mechanised framework presented in Chapter 5 reveal that, for stable compatibility properties such as the termination of a system or the occurrence of an unexpected reception, the compatibility result do not seem to depend on the communication model provided it is *Causal*, *FIFO 1-n*, *FIFO n-1*, or *FIFO n-n*. Given a composition of peers and such a stable property, the compatibility result is the same under all the four models, despite different sets of runs. Ongoing work aims at proving the *Causal* communication model and the *FIFO n-n* communication model are indeed equivalent when it comes to stable compatibility properties. The equivalence between all the four models would derive from the already established hierarchies.

The equivalence between these models would reveal to be instrumental in the compatibility checking of systems because it would allow to substitute a complex communication model such as *Causal* with a formally simple model such as *FIFO n-n* in order to ease proofs of compatibility. In the remote future, looking for other classes of equivalence, for instance among the new kinds of communication models evoked in the previous section, would also help move towards this goal.

Bibliography

- [ABDF08] Ali Ait-Bachir, Marlon Dumas, and Marie-Christine Fauvet. BESERIAL: Behavioural Service Analyser. In *Business Process Management International Conference. Demo session.*, pages 374–377, 2008. LNCS 5240.
- [ABH⁺10] Jean-Raymond Abrial, Michael J. Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *STTT*, 12(6):447–466, 2010.
- [Abr10] Jean-Raymond Abrial. *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, 2010.
- [ACM03] Jean-Raymond Abrial, Dominique Cansell, and Dominique Méry. A mechanically proved and incremental development of IEEE 1394 tree identify protocol. *Formal Aspects of Computing*, 14(3):215–227, 2003.
- [AH07] Jean-Raymond Abrial and Stefan Hallerstede. Refinement, decomposition, and instantiation of discrete models: Application to Event-B. *Fundamenta Informaticae*, 77(1-2):1–28, 2007.
- [AHP96] Rajeev Alur, Gerard J Holzmann, and Doron Peled. An analyzer for message sequence charts. In *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *LNCS*, pages 35–48. Springer, 1996.
- [AMS12] Manamiary Bruno Andriamarina, Dominique Méry, and Neeraj Kumar Singh. Revisiting snapshot algorithms by refinement-based techniques. In *13th International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT 2012*, pages 343–349. IEEE, 2012.
- [AMS14] Manamiary Bruno Andriamarina, Dominique Méry, and Neeraj Kumar Singh. Revisiting snapshot algorithms by refinement-based techniques. *Computer Science and Information Systems*, 11(1):251–270, 2014.
- [AMW17] Noran Azmya, Stephan Merz, and Christoph Weidenbach. A machine-checked correctness proof for Pastry. *Science of Computer Programming*, 2017. To appear.
- [AY99] Rajeev Alur and Mihalis Yannakakis. *Model Checking of Message Sequence Charts*, pages 114–129. Springer Berlin Heidelberg, 1999.
- [BB92] Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96(1):217–248, 1992.

- [BBO12] Samik Basu, Tevfik Bultan, and Meriem Ouederni. Synchronizability for verification of asynchronously communicating systems. In *13th International Conference on Verification, Model Checking, and Abstract Interpretation*, volume 7148 of *LNCS*, pages 56–71. Springer-Verlag, 2012.
- [BCPV04] Antonio Brogi, Carlos Canal, Ernesto Pimentel, and Antonio Vallecillo. Formalizing web service choreographies. *Electronic Notes in Theoretical Computer Science*, 105:73–94, December 2004.
- [BCT04] Boualem Benatallah, Fabio Casati, and Farouk Toumani. Analysis and management of web service protocols. In *Conceptual Modeling – ER 2004*, volume 3288 of *Lecture Notes in Computer Science*, pages 524–541. Springer, 2004.
- [Bir96] Kenneth P. Birman. *Building secure and reliable network applications*. Manning, 1996.
- [BJ87] Kenneth P. Birman and Thomas A. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):47–76, January 1987.
- [BK88] Ralph-Johan Back and Reino Kurki-Suonio. Distributed cooperation with action systems. *ACM Transactions on Programming Languages and Systems*, 10(4):513–554, 1988.
- [BM03] Michel Bauderon and Mohamed Mosbah. A unified framework for designing, implementing and visualizing distributed algorithms. *Electronic Notes in Theoretical Computer Science*, 72(3):13–24, 2003.
- [BM13] Michael Butler and Issam Maamria. Practical theory extension in Event-B. In Zhiming Liu, Jim Woodcock, and Huibiao Zhu, editors, *Theories of Programming and Formal Methods*, volume 8051 of *Lecture Notes in Computer Science*, pages 67–81. Springer Berlin Heidelberg, 2013.
- [Bou92] Gérard Boudol. Asynchrony and the Pi-calculus. Research Report RR-1702, INRIA, 1992.
- [BPV08] Romain Beauxis, Catuscia Palamidessi, and Frank D. Valencia. On the asynchronous nature of the asynchronous π -calculus. In Rocco De Nicola, Pierpaolo Degano, and José Meseguer, editors, *Concurrency, Graphs and Models*, volume 5065 of *Lecture Notes in Computer Science*, pages 473–492. Springer, 2008.
- [Bry11] Jeremy W. Bryans. Developing a consensus algorithm using stepwise refinement. In *13th International Conference on Formal Methods and Software Engineering, ICFEM’11*, pages 553–568. Springer-Verlag, 2011.
- [BTMK16] Maha Boussabbeh, Mohamed Tounsi, Mohamed Mosbah, and Ahmed Hadj Kacem. Formal proofs of termination detection for local computations by refinement-based compositions. In *Proceedings of the 5th International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z - Volume 9675*, ABZ 2016, pages 198–212. Springer-Verlag New York, Inc., 2016.
- [But09] Michael J. Butler. Decomposition structures for Event-B. In *Integrated Formal Methods, 7th International Conference, IFM 2009*, pages 20–38, 2009.

- [BZ83] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, April 1983.
- [CBMT96] Bernadette Charron-Bost, Friedemann Mattern, and Gerard Tel. Synchronous, asynchronous, and causally ordered communication. *Distributed Computing*, 9(4):173–191, February 1996.
- [CDK94] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: concepts and design*. Addison Wesley, second edition, 1994.
- [CDLM10] Kaustuv Chaudhuri, Damien Doligez, Leslie Lamport, and Stephan Merz. Verifying safety properties with the TLA⁺ Proof System. In *Proceedings of the 5th International Conference on Automated Reasoning*, volume 6173 of *LNCS*, pages 142–148. Springer-Verlag, 2010.
- [CF99] Flaviu Cristian and Christof Fetzer. The timed asynchronous distributed system model. *IEEE Transaction on Parallel and Distributed Systems*, 10(6):642–657, June 1999.
- [CG98] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In *First International Conference on Foundations of Software Science and Computation Structure*, FoSSaCS ’98, pages 140–155. Springer-Verlag, 1998.
- [CHMQ16] Florent Chevrou, Aurélie Hurault, Philippe Mauran, and Philippe Quéinnec. Mechanized refinement of communication models with TLA⁺. In *5th Intl. Conf. Abstract State Machines, Alloy, B, TLA, VDM, and Z (ABZ 2016)*, volume 9675 of *LNCS*, pages 312–318. Springer-Verlag, May 2016.
- [CHQ15] Florent Chevrou, Aurélie Hurault, and Philippe Quéinnec. Automated verification of asynchronous communicating systems with TLA⁺. *Electronic Communications of the EASST (PostProceedings of the 15th International Workshop on Automated Verification of Critical Systems)*, 72, 2015.
- [CHQ16] Florent Chevrou, Aurélie Hurault, and Philippe Quéinnec. On the diversity of asynchronous communication. *Formal Aspects of Computing*, 28(5):847–879, September 2016.
- [CLB08] Heung Seok Chae, Joon-Sang Lee, and Jung Ho Bae. An approach to checking behavioral compatibility between web services. *International Journal of Software Engineering and Knowledge Engineering*, 18(2):223–241, 2008.
- [CM88] K. Mani Chandy and Jayadev Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.
- [CPS93] Rance Cleaveland, Joachim Parrow, and Bernhard Steffen. The concurrency workbench: A semantics-based tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, January 1993.
- [CPT01] Carlos Canal, Ernesto Pimentel, and José M. Troya. Compatibility and inheritance in software architectures. *Science of Computer Programming*, 41(2):105–138, October 2001.
- [Dij83] Edsger W. Dijkstra. EWD851b – reducing control traffic in a distributed implementation of mutual exclusion, 1983.

- [DOS12] Francisco Durán, Meriem Ouederni, and Gwen Salaün. A generic framework for n-protocol compatibility checking. *Science of Computer Programming*, 77(7-8):870–886, July 2012.
- [DS90] Edsger W. Dijkstra and Carel S. Scholten. *Predicate Calculus and Program Semantics*. Springer-Verlag New York, Inc., 1990.
- [DSU04] Xavier Défago, André Schiper, and Péter Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys*, 36:372–421, December 2004.
- [DWZ⁺06] Shuiguang Deng, Zhaohui Wu, Mengchu Zhou, Ying Li, and Jian Wu. Modeling service compatibility with pi-calculus for choreography. In *25th International Conference on Conceptual Modeling*, Conceptual Modeling - ER 2006, pages 26–39. Springer-Verlag, 2006.
- [EMR02] André Engels, Sjouke Mauw, and Michel A. Reniers. A hierarchy of communication models for message sequence charts. *Science of Computer Programming*, 44(3):253–292, 2002.
- [FG96] Cédric Fournet and Georges Gonthier. The reflexive cham and the join-calculus. In *23rd ACM Symposium on Principles of Programming Languages*, POPL ’96, pages 372–385. ACM, 1996.
- [Fid88] Colin J. Fidge. Timestamps in message-passing systems that preserve the partial ordering. In *11th Australian Computer Science Conference*, pages 55–66, February 1988.
- [FUMK04] Howard Foster, Sebastian Uchitel, Jeff Magee, and Jeff Kramer. Compatibility verification for web service choreography. In *IEEE International Conference on Web Services*, pages 738–, 2004.
- [GGH⁺10] Paul Gardiner, Michael Goldsmith, Jason Hulance, David Jackson, Bill Roscoe, Bryan Scattergood, and Philip Armstrong. FDR2 user manual. Technical report, Oxford University, november 2010.
- [GLMS13] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. CADP 2011: a toolbox for the construction and analysis of distributed processes. *International Journal on Software Tools for Technology Transfer*, 15(2):89–107, 2013.
- [HHK⁺15] Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jay Lorch, Bryan Parno, Michael Lowell Roberts, Srinath Setty, and Brian Zill. Ironfleet: Proving practical distributed systems correct. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, October 2015.
- [HISW11] Thai Son Hoang, Alexei Iliasov, Renato Silva, and Wei Wei. A survey on Event-B decomposition. *ECEASST*, 46, 2011.
- [Hoa78] Charles Antony Richard Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978.
- [Hol04] Gerard J. Holzmann. *The Spin Model Checker : Primer and Reference Manual*. Addison-Wesley, 2004.

- [HT91] Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In *Proceedings of the European Conference on Object-Oriented Programming, ECOOP '91*, pages 133–147. Springer-Verlag, 1991.
- [ILTR11] Alexei Iliasov, Linas Laibinis, Elena Troubitsyna, and Alexander Romanovsky. Formal derivation of a distributed program in Event-B. In *ICFEM*, volume 6991 of *Lecture Notes in Computer Science*, pages 420–436. Springer, 2011.
- [ISO89] ISO. Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour. ISO 8807:1989, International Organization for Standardization, 1989.
- [ISO01] ISO. Information technology – Enhancements to LOTOS (E-LOTOS). ISO 15437:2001, International Organization for Standardization, 2001.
- [ITL⁺10] Alexei Iliasov, Elena Troubitsyna, Linas Laibinis, Alexander Romanovsky, Kimmo Varpaaniemi, Dubravka Ilic, and Timo Latvala. Supporting reuse in Event B development: Modularisation approach. In *Abstract State Machines, Alloy, B and Z*, pages 174–188, 2010.
- [Jos92] Mark B. Josephs. Receptive process theory. *Acta Informatica*, 29(1):17–31, 1992.
- [KAB⁺07] Charles Edwin Killian, James W. Anderson, Ryan Braud, Ranjit Jhala, and Amin M. Vahdat. Mace: Language support for building distributed systems. *SIGPLAN Not.*, 42(6):179–188, June 2007.
- [KAJV07] Charles Edwin Killian, James W. Anderson, Ranjit Jhala, and Amin Vahdat. Life, death, and the critical transition: Finding liveness bugs in systems code (awarded best paper). In *4th Symposium on Networked Systems Design and Implementation (NSDI 2007), April 11-13, 2007, Cambridge, Massachusetts, USA, Proceedings.*, 2007.
- [KS98] Ajay D. Kshemkalyani and Mukesh Singhal. Necessary and sufficient conditions on information for causal message ordering and their optimal implementation. *Distributed Computing*, 11(2):91–111, 1998.
- [KS11] Ajay D. Kshemkalyani and Mukesh Singhal. *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, March 2011.
- [Lam78] Leslie Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.
- [Lam94] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994.
- [Lam01] Butler W. Lampson. The ABCD’s of Paxos. In *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing, PODC 2001*, pages 13–. ACM, 2001.
- [Lam02] Leslie Lamport. *Specifying Systems*. Addison Wesley, 2002.
- [Lam09] Leslie Lamport. The PlusCal algorithm language. In Martin Leucker and Carroll Morgan, editors, *Theoretical Aspects of Computing*, volume 5684 of *Lecture Notes in Computer Science*, pages 36–60. Springer, 2009.

- [Lam10] Leslie Lamport. Mechanically checked safety proof of a byzantine Paxos algorithm. <http://research.microsoft.com/en-us/um/people/lamport/tla/byzpaxos.html>, 2010.
- [Lam11] Leslie Lamport. Byzantizing paxos by refinement. In David Peleg, editor, *25th International Symposium on Distributed Computing, DISC 2011*, volume 6950 of *Lecture Notes in Computer Science*, pages 211–224. Springer, 2011.
- [Lei10] K Rustan M Leino. Dafny: An automatic program verifier for functional correctness. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, volume 6355 of *LNCS*, pages 348–370. Springer, 2010.
- [LFS⁺11] Xitong Li, Yushun Fan, Q. Z. Sheng, Z. Maamar, and Hongwei Zhu. A Petri net approach to analyzing behavioral compatibility and similarity of web services. *IEEE Transactions on Systems, Man and Cybernetics*, 41(3):510–521, May 2011.
- [LMP04] Xiaozhou Li, Jayadev Misra, and C. Greg Plaxton. Active and concurrent topology maintenance. In Rachid Guerraoui, editor, *Distributed Computing, 18th International Conference*, volume 3274 of *Lecture Notes in Computer Science*, pages 320–334. Springer, 2004.
- [LMW11] Tianxiang Lu, Stephan Merz, and Christoph Weidenbach. Towards verification of the Pastry protocol using TLA⁺. In Roberto Bruni and Jürgen Dingel, editors, *International Conference on Formal Techniques for Distributed Systems FORTE*, volume 6722 of *Lecture Notes in Computer Science*, pages 244–258. Springer, 2011.
- [Lon12] Delphine Longuet. Global and local testing from message sequence charts. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC '12*, pages 1332–1338. ACM, 2012.
- [Lu13] Tianxiang Lu. *Formal Verification of the Pastry Protocol*. PhD thesis, Université de Lorraine – Universität des Saarlandes, July 2013.
- [Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., 1996.
- [Mar03] Axel Martens. On compatibility of web services. *Petri Net Newsletter*, pages 12–20, 2003.
- [Mat89] Friedemann Mattern. Virtual time and global state in distributed systems. In *Int'l Workshop on Parallel and Distributed Algorithms*, pages 215–226. Elsevier Science Publishers, 1989.
- [Mil82] Robin Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., 1982.
- [Mil99] Robin Milner. *Communicating and Mobile Systems: The π -calculus*. Cambridge University Press, 1999.
- [MR94] Sjouke Mauw and Michel A. Reniers. An algebraic semantics of basic message sequence charts. *The computer journal*, 37(4):269–277, 1994.
- [Mul93] Sape J. Mullender, editor. *Distributed Systems*. ACM Press Frontier Series, second edition, 1993.

- [OSB13] Meriem Ouederni, Gwen Salaün, and Tevfik Bultan. Compatibility checking for asynchronously communicating software. In *International Symposium on Formal Aspects of Component Software (FACS 2013)*, volume 8348 of *LNCS*, pages 310–328, 2013.
- [Pal03] Catuscia Palamidessi. Comparing the Expressive Power of the Synchronous and the Asynchronous pi-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.
- [PBS89] Larry L. Peterson, Nick C. Buchholz, and Richard D. Schlichting. Preserving and using context information in interprocess communication. *ACM Transactions on Computer Systems*, 7(3):217–246, 1989.
- [Pra91] K. V. S. Prasad. A calculus of broadcasting systems. In S. Abramsky and T. S. E. Maibaum, editors, *TAPSOFT '91: Proceedings of the International Joint Conference on Theory and Practice of Software Development Brighton, UK, April 8–12, 1991*, volume 493 of *LNCS*, pages 338–358. Springer, 1991.
- [PRS97] Ravi Prakash, Michel Raynal, and Mukesh Singhal. An adaptive causal ordering algorithm suited to mobile computing environments. *Journal of Parallel and Distributed Computing*, 41(2):190–204, March 1997.
- [Ray10] Michel Raynal. *Communication and Agreement Abstractions for Fault-tolerant Asynchronous Distributed Systems*. Synthesis Lectures on Distributed Computing Theory. Morgan and Claypool Publishers, 2010.
- [Ray13] Michel Raynal. *Distributed Algorithms for Message-Passing Systems*. Springer, 2013.
- [RRJ⁺14] Daniel Ricketts, Valentin Robert, Dongseok Jang, Zachary Tatlock, and Sorin Lerner. Automating formal proofs for reactive systems. *SIGPLAN Not.*, 49(6):452–462, June 2014.
- [RRM05] John Risson, Ken Robinson, and Tim Moors. Fault tolerant active rings for structured peer-to-peer overlays. In *IEEE Conference on Local Computer Networks*, LCN '05, pages 18–25. IEEE Computer Society, 2005.
- [RST91] Michel Raynal, André Schiper, and Sam Toueg. The causal ordering abstraction and a simple way to implement it. *Information Processing Letters*, 39:343–350, October 1991.
- [SHQ17] Nathanael Sensfelder, Aurélie Hurault, and Philippe Quéinnec. Inference of Channel Priorities for Asynchronous Communication (regular paper). In *International Conference on Distributed Computing and Artificial Intelligence*, volume 620 of *Advances in Intelligent Systems and Computing*. Springer, 2017.
- [Sil11] Renato Silva. Towards the composition of specifications in Event-B. *Electronic Notes in Theoretical Computer Science*, 280:81–93, 2011.
- [SM94] Reinhard Schwarz and Friedemann Mattern. Detecting causal relationships in distributed computations: In search of the holy grail. *Distributed Computing*, 7(3):149–174, June 1994.
- [SY10] Raghuraj Suryavanshi and Divakar Yadav. Formal development of byzantine immune total order broadcast system using Event-B. In *ICDEM*, volume 6411 of *Lecture Notes in Computer Science*, pages 317–324. Springer, 2010.

- [Tel00] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, second edition, 2000.
- [TFZ09] Wei Tan, Yushun Fan, and MengChu Zhou. A Petri net-based method for compatibility analysis and composition of web services in business process execution language. *IEEE Transactions on Automation Science and Engineering*, 6(1):94–106, 2009.
- [TMM16] Mohamed Tounsi, Mohamed Mosbah, and Dominique Méry. From Event-B specifications to programs for distributed algorithms. *Int. J. Auton. Adapt. Commun. Syst.*, 9(3/4):223–242, January 2016.
- [WWP⁺15] James R. Wilcox, Doug Woos, Pavel Panchekha, Zachary Tatlock, Xi Wang, Michael D. Ernst, and Thomas Anderson. Verdi: A framework for implementing and formally verifying distributed system. In *PLDI 2015, Proceedings of the ACM SIGPLAN 2015 Conference on Programming Language Design and Implementation*, June 15–17, 2015.
- [YB05] Divakar Yadav and Michael J. Butler. Application of Event B to global causal ordering for fault tolerant transactions. In *Workshop on Rigorous Engineering of Fault Tolerant Systems (REFT2005)*, pages 93–102, July 2005.
- [YKKK09] Maysam Yabandeh, Nikola Knezevic, Dejan Kostic, and Viktor Kuncak. Crystal-ball: Predicting and preventing inconsistencies in deployed distributed systems. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, NSDI’09, pages 229–244. USENIX Association, 2009.
- [Zav12] Pamela Zave. Using lightweight modeling to understand Chord. *SIGCOMM Computer Communication Review*, 42(2):49–57, April 2012.